UNIVERSITÉ DE MONTRÉAL

RÉALISATION DE LA MÉMOIRE PARTAGÉE DANS LES SYSTÈMES RÉPARTIS

par

Hugues NOTOUOM

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET DE GÉNIE INFORMATIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES (M.Sc.A)

(GÉNIE GÉLECTRIQUE)

JUILLET 1996

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

RÉALISATION DE LA MÉMOIRE PARTAGÉE DANS LES SYSTÈMES RÉPARTIS

Présenté par: Hugues NOTOUOM
en vue de l'obtention du diplôme de: Maîtrise Ès Sciences Appliquées
a été dûment accepté par le jury d'examen constitué de:

M. Jean-Louis HOULE , Ph.D., président
M. Claude ÉVÉQUOZ , Ph.D., membre et directeur de recherche
M. Louis GRANGER, M.Sc., membre et substitut du directeur de recherche
M. Hai Hoc Hoang , Ph.D., membre

Dédicace

À ma feue grand-mère, Madame Méwa Odette, en souhaitant que son âme se repose en paix.

REMERCIEMENTS

L'auteur tient à remercier tous ceux qui ont participé de près ou de loin à la réalisation de ce mémoire.

Je remercie surtout mon directeur de recherche Monsieur Claude Évéquoz, pour les idées qu'il a apportées pour la réalisation de ce projet ainsi que le temps qu'il a consacré pour lire et corriger la version finale et les versions préliminaires de ce document.

RÉSUMÉ

Pour augmenter la puissance de calcul, on utilise de plus en plus les systèmes répartis. On constate cependant que l'absence d'une mémoire commune les rendent difficile à programmer, car le paradigme de programmation par messages qu'ils offrent à l'usager est différent du paradigme de programmation qui lui est offert par les machines mono-processeurs et auxquelles ils est habitué.

L'objectif principal de ce projet est triple:

- implanter dans les systèmes répartis et par une approche serveur, une mémoire partagée et globale en utilisant les modules mémoires disponibles à chaque poste de travail;
- proposer un modèle analytique pour évaluer rapidement les performances du système résultant;
- étudier l'impact des paramètres sur ce dernier.

Ce mémoire décrit donc une approche vers la réalisation de ces objectifs.

Le système que nous avons retenu est composé de postes de travail, de serveurs principaux et de serveurs secondaires, tous reliés par un réseau de communications à grand débit. La cohérence entre les données partagées est maintenue d'une part par un protocole de mise à jour en écriture, et de l'autre par un protocole d'invalidation en écriture. Ces deux protocoles utilisent le schéma du propriétaire de la donnée ainsi que le schéma de la copie initiale. Pour transmettre une information locale à plusieurs sites, le système utilise une des trois techniques suivantes: la diffusion, la diffusion simulée et le chaînage. Ces techniques de diffusions sont couplées à un mécanisme de verrouillage à deux phases pour s'assurer que la donnée propagée est correctement reçue par tous les sites participants.

Pour chaque protocole utilisé, un modèle analytique hiérarchisé a été développé. Ce modèle est composé d'une chaîne de Markov de l'état des données, d'une chaîne de Markov du nombre de copies des données, ainsi que d'un modèle du processeur formé d'un réseau mixte de files d'attente ouvertes et fermées. Dans chaque cas, un modèle de simulation a également été proposé afin de valider le modèle analytique. Une des caractéristiques de chaque modèle obtenu est qu'il tient compte de l'effet du protocole de cohérence, de la technique de diffusion ainsi que de chaque paramètre sur les performances du système. Pour réduire la complexité du modèle du processeur, un modèle équivalent formé de plusieurs chaînes de routages ouvertes et d'une seule chaîne de routage fermée a été proposé.

Pour chaque modèle analytique obtenu, un programme de mesure de performance a été implanté. L'avantage de ce programme est qu'il ne nécessite pas beaucoup d'espace mémoire, et qu'il donnent un temps de réponse rapide sur un ordinateur personnel, et ceci, quel que soit la taille du système. D'autre part, il permet d'étudier le rôle et la contribution de chaque paramètre sur les performances du système.

Les résultats numériques montrent que pour implanter une mémoire partagée dans les système répartis en utilisant notre approche, il faut que le réseau de communication soit très rapide. Dans ce cas, il faut maximiser le nombre de serveurs principaux et minimiser le nombre de serveurs secondaires afin d'obtenir les meilleurs performances, et d'annuler l'effet du protocole de cohérence ainsi que l'effet la technique de diffusion. Si le réseau n'est pas rapide, il est préférable d'utiliser le protocole de mise à jour couplé à la technique de diffusion.

ABSTRACT

Distributed systems are used more and more to increase computation power. However, they have no common memory. This lack of memory makes them difficult to program, because the message communication paradigm they offer to the user is different from the shared memory communication paradigm with which he is familiarized.

The main objective of our research is triple:

- to build a distributed shared memory using memories modules available in any workstation by a server approach;
- to develop an analytic model to evaluate rapidly the performance of the resulting system;
- to study the impact of the parameters on the performance of the system.

This thesis presents an approach to reach this goal.

The system consists of workstations, main and backup servers. This system can use either write-update protocol, or write-invalidate protocol. It can also support true or simulated broadcast, or chain-bond communications. To make sure that data are correctly propagated, these techniques are coupled to a weak from of consistency by two-phase locking.

For each of these protocols and for both broadcast communications, an analytic model is develop. It is composed of a Markov chain representing the states of data in the local memory, a Markov chain representing the number of copies of data in the system, and a processor model. This processor model consists of mixed multiple chain queueing networks. A simulation model is developed to validate the analytic model. To reduce the complexity of the processor model, an equivalent model is proposed. It is composed of a single closed chain and of multiple open chain queueing networks.

For each model, a computer program is wrote. The advantage of these programs is that they quickly give results on personal computers. Furthermore, it facilitate the study of the impact of the parameters on the system performance.

Numerical results obtained from the model show that, to build a shared memory on top of a distributed system using our approach, one needs a high bandwidth interconnection network. In this case, the number of main servers has to be maximized and the number of backup servers minimized in order to obtain better performance and cancel the effect of the coherence protocol and also the broadcast communication. If the network is not fast enough, the write-update protocol should be used with the true broadcast.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION

Un ensemble d'ordinateurs forment un système réparti lorsqu'ils peuvent s'échanger de l'information à l'aide d'un système de communication. Selon le degré de couplage, on distingue les multiprocesseurs dont la communication se réalise généralement par l'intermédiaire d'une mémoire partagée et les réseaux d'ordinateurs. C'est à ces derniers types de système que nous restreignons tout au long de ce mémoire, le qualificatif de réparti.

Depuis l'avènement des ordinateurs, leurs vitesses n'ont jamais cessé d'augmenter pour satisfaire les exigences des applications pour lesquelles ils étaient utilisés. Cependant, avec la limitation physique imposée par la vitesse de la lumière, il est impossible de croire que cette vitesse va croître indéfiniment. Aujourd'hui, on note l'apparition de nouvelles applications ayant des contraintes temporelles d'exécution alors que les performances des ordinateurs saturent. Pour rencontrer les exigences imposées par ces applications, il faut augmenter la puissance de calcul en utilisant plusieurs processeurs en parallèle.

Deux genres de machines parallèles sont devenus populaires: les multiprocesseurs et les systèmes répartis. Les multiprocesseurs sont constitués de plusieurs processeurs et d'une seule mémoire physique globale. L'avantage de ces systèmes est qu'ils sont faciles à programmer étant donné qu'ils sont naturellement assimilables aux machines monoprocesseurs. Cependant, leurs performances sont affectées par le goulot d'étranglement qui apparait au niveau de la mémoire principale lorsque le nombre de processeurs augmente. En effet, la mémoire ne dispose en général que d'un seul port

d'accès et par conséquent traite les requêtes séquentiellement. Cette faiblesse limite le nombre de processeurs qu'on peut coupler à la mémoire et réduit l'efficacité du traitement parallèle. Les systèmes répartis, quant à eux, ne souffrent pas des mêmes problèmes.

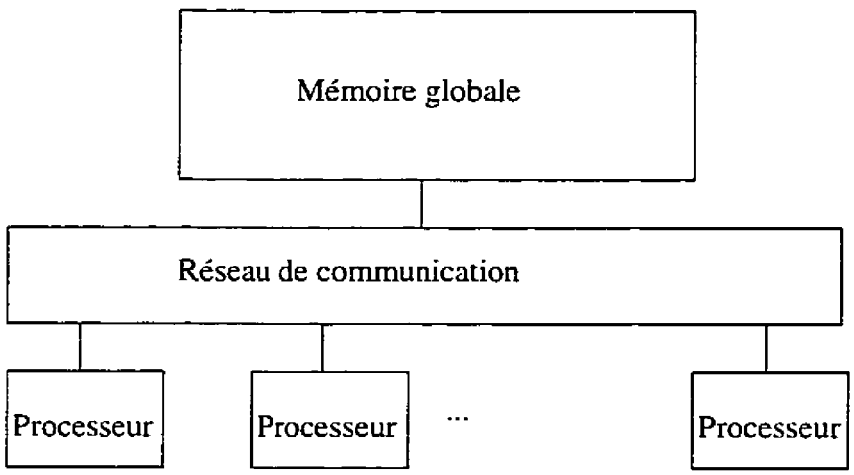


Figure 1.1 Un système multiprocesseur

Avec la chute du prix des processeurs et des mémoires, jumelée à l'accroissement de leurs performances, on a assisté à la pénétration de l'informatique dans plusieurs secteurs d'activité sous la forme de micro-ordinateurs. Pour plusieurs applications, un terminal d'accès à un système à temps partagé peut être remplacé avantageusement par un micro-ordinateur. Cependant, il faut noter que:

- la communication entre usagers et le partage des ressources que permettaient les systèmes à temps partagé ne sont plus possibles avec des micro-ordinateurs isolés.

Ces problèmes justifient le développement des systèmes répartis interconnectant des postes de travail individuels et des serveurs spécialisés à travers un réseau de communication à grand débit. Si la topologie du réseau est bien conçue, ces systèmes peuvent contenir plus de sites que les multiprocesseurs.

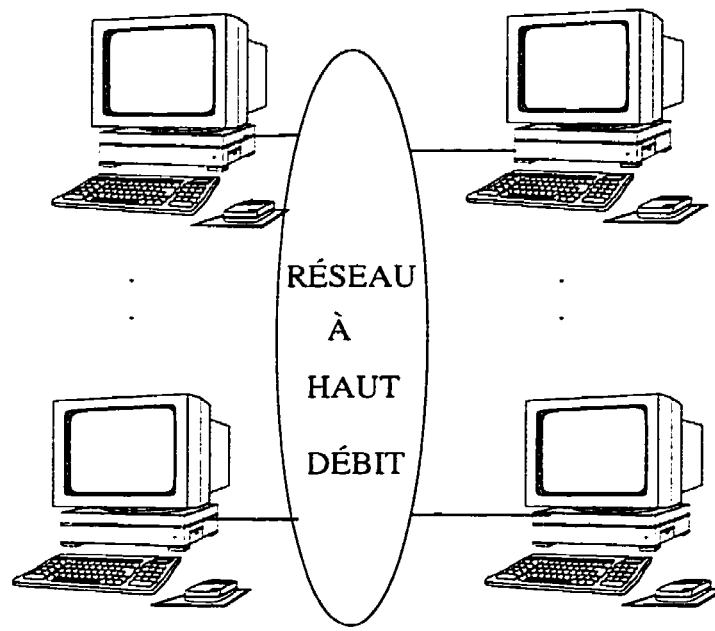


Figure 1.2 Un système réparti

Un système informatique réparti doit permettre le partage des ressources, l'autonomie et la disponibilité de ses composants, une extensibilité ainsi qu'une hétérogénéité.

Le partage des ressources: Dans un système réparti, certaines ressources peuvent être jugées très coûteuses ou sous-utilisées. Si plusieurs postes de travail se partagent les mêmes ressources, alors on augmente leur taux d'utilisation et on réduit en même temps le

coût associé au système. Ce partage permet aussi la répartition de la charge par la migration des données.

L'autonomie: Ceci permet d'interconnecter des applications déjà existantes, développées de manière indépendante pour former un système unique intégré. Le fonctionnement des postes est indépendant les uns des autres mais ceux-ci peuvent communiquer entre eux.

La disponibilité: Le mauvais fonctionnement d'un composant ne doit pas entraîner la panne du système. Celui-ci doit être capable de détecter un composant défectueux pour permettre son remplacement par un composant équivalent. Il faut donc prévoir un logiciel pour gérer la reconfiguration ou la reprise en cas d'incident. C'est à dire que le système doit être tolérant aux fautes ou aux défaillances.

L'extensibilité: Un système réparti est toujours appelé à grossir. Ainsi le système doit supporter en tout temps, l'ajout de nouvelles applications et de nouveaux composants sans pour autant affecter la qualité de son service.

L'hétérogénéité: Plusieurs équipements d'origines différentes doivent cohabiter au sein du même système. Étant donné que ces composants n'ont pas les mêmes normes, il faut prévoir un mécanisme permettant de les uniformiser.

1.1 Évolution des systèmes répartis

Vers la fin des années 50, on notait l'apparition des premiers systèmes répartis, notamment le système SABRE utilisé pour la réservation des places d'avion ou le système SAGE

servant à la surveillance du territoire [Krakowiak 87]. La caractéristique commune de ces systèmes était qu'ils étaient très spécialisés et utilisaient tous la technique de communication point à point pour l'échange d'information. Par la suite les efforts ont été orientés vers la conception des systèmes répartis très généraux. Un des premiers projets d'envergure a été ARPANET qui fut lancé au début des années 60 [Bertekas 89]. Ces projets ont permis de mettre au point la technique de commutation par paquet et de développer des protocoles de communication, notamment pour le transfert de fichiers.

Au cours des années 70, on note l'élaboration du principe du réseau local ETHERNET, des réseaux à anneau à jeton et de l'organisation client-serveur pour le partage des ressources. Ces principes ont permis au cours des années 80 de développer des systèmes répartis intégrés.

Avec l'introduction des réseaux à fibre optique qui présentent l'avantage d'offrir des taux de transmission élevés et une transmission sans erreur, les projets actuels s'orientent vers le développement des systèmes répartis pouvant gérer des applications qui nécessitent une grande largeur de bande comme le multimédia.

1.2 Caractérisriques des systèmes répartis

Un système réparti doit être muni d'un système d'exploitation. Ce système d'exploitation doit être vu par l'usager comme étant un système d'exploitation ordinaire qui fonctionne dans plusieurs processeurs; c'est à dire que le système d'exploitation doit être transparent. En d'autres termes, l'utilisation de plusieurs processeurs doit être invisible par l'usager. Celui-ci voit le système comme une machine monoprocesseur virtuelle et pas comme une

collection de machines distinctes.

Les systèmes répartis présentent plusieurs attraits. Premièrement, le gain en performance de processeurs associé à la chute de leurs prix font que leurs rapports prix/performance est avantageux par rapport à certains systèmes traditionnels tels que les mini-ordinateurs. Un autre facteur qui joue en faveur des systèmes répartis est leur extensibilité. Si on veut accroître la puissance de calcul d'un certain pourcentage, il suffit d'augmenter le nombre de processeurs par la même proportion. Enfin, les systèmes répartis offrent une très grande disponibilité. Si une partie du système tombe en panne, les autres ne sont pas trop affectées. Cependant, l'efficacité du système en est affectée car le nombre de processeurs à allouer est réduit.

Plusieurs modèles ont été proposés pour construire les systèmes répartis. Ces modèles peuvent être classés en trois catégories: le modèle à mini-ordinateurs, le modèle à postes de travail et le modèle à banque de processeurs [Tanenbaum 85].

Dans le modèle à mini-ordinateurs, le système est constitué d'environ une douzaine de mini-ordinateurs (VAX par exemple) chacun ayant plusieurs clients. Chaque client se connecte à une machine spécifique avec un accès sur les autres. Ce modèle n'est qu'une extension du modèle à temps partagé.

Dans le modèle à postes de travail, chaque client possède une machine comportant un processeur puissant, une mémoire, un écran, et un disque. Tout le travail s'effectue au niveau du poste de travail. Un tel système commence à devenir réparti lorsqu'il dispose d'un système de fichiers global unique et où les données sont obtenues sans préciser leur

endroit d'origine. Autrement dit, l'accès aux données est transparent pour l'utilisateur.

Le modèle à banque de processeurs est la prochaine étape d'évolution après le modèle à postes de travail. Si un client veut une certaine puissance de calcul, un ou plusieurs processeurs peuvent lui être alloués temporairement. Dès que le travail se termine, les processeurs sont remis dans la banque et attendent leur prochaine utilisation. Un modèle hybride peut être conçu en combinant ce modèle à celui des postes de travail. Ainsi, chaque utilisateur dispose d'une banque de plusieurs processeurs pour ses calculs intensifs en plus de son propre poste de travail.

1.3 Mémoire partagée

Les systèmes répartis présentent un degré de parallélisme. Malgré qu'ils ne disposent pas de mémoire commune, ils peuvent néanmoins exécuter des programmes parallèles. Dans ces applications, les programmes créent des processus qui manipulent des données qu'ils partagent. L'accès à une variable partagée se fait par l'échange de messages. Ces systèmes sont difficiles à programmer car le modèle de programmation est basé sur le paradigme de communication par message, technique qui est différente du paradigme de programmation par mémoire partagée utilisé dans les langages procéduraux auxquels le programmeur est habitué. Pour faciliter la programmation, certains systèmes ont implanté une mémoire commune virtuelle dans les systèmes répartis. L'idée est de donner au programmeur l'illusion que le système dispose d'une mémoire partagée et de lui permettre ainsi d'utiliser le paradigme de programmation par mémoire partagée.

L'implantation d'une mémoire partagée se fait en deux étapes:

- la spécification physique;
- la spécification du fonctionnement logique.

1.3.1 La spécification physique

Dans cette première étape, il faut d'abord déterminer la répartition physique de la mémoire partagée. Deux méthodes sont généralement utilisées pour implanter physiquement une mémoire globale: la manière répartie où l'espace adressable est local et l'accès à une variable globale se fait par l'émission de messages à travers le réseau, et la manière partagée où l'espace adressable comprend toutes les mémoires du système. Avec la première approche, le système peut supporter un nombre important de composants mais il devient difficile à programmer pour les raisons que nous avons énumérées auparavant. Avec la deuxième approche, les systèmes sont faciles à programmer mais au point de vue architecturale, ils sont inefficaces car lorsque le système devient grand, la mémoire partagée devient un goulot d'étranglement. Une manière de réunir les deux paradigmes consiste à utiliser l'avantage de chacun: il s'agit d'une mémoire physiquement répartie mais à espace adressable partagé. Par la suite on spécifie la structure et l'organisation de la mémoire ainsi que de la taille des objets manipulés et leurs représentations internes.

1.3.2 Fonctionnement logique

Dans cette deuxième étape, on décrit comment les données partagées sont gérées par le système. Deux approches sont généralement utilisées: la méthode statique et la méthode dynamique [Nitzberg 91]. Pour le premier cas, les données sont disponibles à un endroit précis et les localiser devient facile. Cependant, l'efficacité du système dépend de la

répartition des données partagées. Dans ce cas, il est possible que toutes les données se retrouvent à un seul site créant ainsi un goulot d'étranglement. Pour le second cas, on a une répartition des données là où elles ont été utilisées. Cette approche permet donc une migration des données dans le système. Dans ce dernier cas, localiser une donnée devient plus difficile et requiert un algorithme. Si l'algorithme est bien conçu, les données seront équitablement réparties et la charge ainsi que les performances du système seront meilleures que dans le cas précédent.

Pour augmenter la disponibilité des données, un système réparti doit permettre leurs replications: c'est-à- dire conserver plusieurs copies d'une donnée dans le système. Ainsi plusieurs postes différents peuvent accéder en même temps à la même donnée. Ceci peut créer des problèmes d'incohérence entre les copies d'une donnée. Supposons que nous disposons de deux copies de la variable X dans deux sites p et q. Si le site p modifie la valeur de X, alors deux valeurs distinctes se retrouvent dans le système: la variable est incohérente. Or si la valeur de X au site q est invalidée ou mise à jour, alors X redevient cohérente. Pour que le système fonctionne bien, il faut donc prévoir un mécanisme pour maintenir la cohérence entre les données. Il s'agit donc de définir un protocole de cohérence.

L'une des deux familles de protocoles suivantes est utilisée pour maintenir la cohérence entre les données partagées: le protocole d'invalidation en écriture et le protocole de mise à jour en écriture [Stenström 90]. Ces deux familles de protocoles fonctionnent identiquement lorsqu'il s'agit de lire une donnée. Par contre, lorsqu'il faut modifier une donnée, les mécanismes diffèrent. Pour ce qui est du protocole d'invalidation en écriture, il faut invalider toutes les autres copies sauf celle qu'il faut modifier. Pour ce qui est du

protocole de mise à jour en écriture, il faut modifier toutes les copies de la donnée dans le système.

Il faut par la suite définir un mécanisme de synchronisation pour empêcher une incohérence entre les données lors d'un accès concurrent. En effet, considérons le cas où une opération d'écriture doit mettre à jour toutes les autres copies du système. Si nous autorisons une opération de lecture sur une des copies précédentes avant la fin des mises à jour, alors l'initiateur de la requête travaillera avec une copie périmée. D'où l'importance d'établir un ordre lorsque l'on veut opérer sur les données partagées. En dernier, on spécifie l'algorithme de traitement lors d'une faute d'objets c'est-à-dire l'ensemble des procédures à suivre pour obtenir une donnée qui est localement absente.

L'objectif principal de ce mémoire est de déterminer les bénéfices et les mérites de ces familles de protocoles. Pour réaliser ce but, nous avons recours à l'analyse de performance que nous introduisons brièvement ci-dessous.

1.4 Analyse de performance

Pour évaluer les performances des systèmes informatiques, on effectue généralement une modélisation analytique du système ou un modèle de simulation.

Pendant la phase de conception, ce modèle analytique permet:

- d'avoir les caractéristiques globales du système;
- d'obtenir la mesure de certains paramètres critiques;
- de déterminer des éventuels goulots d'étranglement;

- de caractériser le système sous certaines conditions d'utilisation.

L'usage de tels modèles est généralement économique en temps d'exécution car il permet d'obtenir rapidement les paramètres recherchés. Cependant, lorsqu'on souhaite une plus grande précision, ou que les hypothèses des modèles ne sont pas satisfaites, on doit avoir recours à la simulation. Dans le cadre de ce mémoire, nous utilisons la simulation pour valider les modèles analytiques que nous avons développés.

Omis les considérations économiques, la simulation est plus complète par rapport à la modélisation analytique en ce sens qu'elle représente fidèlement le comportement du système modélisé dans ses moindres détails. Les résultats produits par le modèle analytique ne sont généralement valable qu'en régime permanent alors que la simulation fournit en plus des résultats en régime transitoire.

1.5 Contributions

Le travail effectué dans le cadre ce mémoire est indispensable pendant la phase de conception d'une mémoire partagée répartie. Ses contributions sont diverses.

Qualitativement, la démarche proposée est indépendante du protocole et elle est valable pour l'évaluation des performances de tous systèmes informatiques. Cette démarche est la suivante:

- Dans un premier temps, extraire du système un premier modèle mettant en jeux les activités de toutes les ressources physiques ainsi que de la façon dont elles sont sollicitées.

- Deuxièmement, obtenir un second modèle mettant en jeu les données sur lesquelles les ressources physiques opèrent.
- Troisièmement, établir la relation entre les deux modèles pour exprimer la dynamique du système et obtenir ainsi un modèle analytique global.
- Quatrièmement, résoudre le modèle analytique obtenu.

Quantitativement, la solution proposée converge rapidement. Elle fournit les paramètres caractéristiques du modèle physique. Ces paramètres guident les concepteurs sur:

- le choix du protocole à utiliser;
- le choix des ressources physiques du système;
- le choix de la taille du système.

Ce qui fait la force de la solution proposée est qu'elle se comporte très bien lorsque la taille du système augmente, ne consomme pas beaucoup d'espace mémoire et en plus fournit un temps de réponse raisonnable sur un ordinateur personnel.

1.6 Contenu du mémoire

Ce chapitre nous a permis d'introduire la problématique du partage de mémoire dans les systèmes répartis. Le chapitre suivant sera consacré à une revue de la littérature. Dans ce chapitre nous ferons une présentation ainsi qu'une analyse des résultats et des algorithmes qui sont utilisés dans les systèmes existantes. Nous le terminerons par un exposé de la méthodologie utilisé par le passé pour obtenir les performances de ces systèmes. Ce chapitre nous sera très utile car il nous permettra d'aborder le chapitre 3 avec la connaissance de tous les paramètres à retenir dans notre modèle. Le chapitre 3 sera donc consacré à la description des modèles sur lesquelles nous allons travailler dans le reste du

mémoire. Ce chapitre présentera d'abord le modèle physique retenu. Par la suite, nous allons décrire les différents protocoles retenus pour assurer son bon fonctionnement. Le chapitre 4 sera consacré à la modélisation. Dans un premier temps des modèles analytiques seront obtenus à partir du système physique et des différents protocoles en utilisant une combinaison de réseau de files d'attente et de chaînes de Markov, ainsi qu'une méthode pour les résoudre. Par la suite des modèles de simulation seront développés pour valider les modèles analytiques précédents. Les résultats obtenus à l'aide de ces différents modèles seront présentés au chapitre 5. Enfin le chapitre 6 présentera les conclusions de ce travail.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

Dans ce chapitre, nous donnons un aperçu des résultats et des algorithmes qui ont servis par le passé à concevoir et à implanter des systèmes à mémoire partagée et répartie (DSM). Nous nous attarderons aussi sur les systèmes qui illustrent le mieux les aspects spécifiques des systèmes à DSM.

Un système réparti est conçu pour exécuter des programmes parallèles. Pour que le système soit efficace, il faut favoriser l'accès simultané aux données, éviter le plus possible les traitements séquentiels et empêcher l'apparition des goulots d'étranglement. Dans les systèmes répartis, les facteurs qui participent à la dégradation du système sont les accès à distance aux données et les opérations de synchronisation. L'effet de ces opérations peut être réduit lors de la phase de conception en faisant un choix judicieux des paramètres qui les influencent ainsi que des algorithmes qui assurent le fonctionnement du système. Ces choix sont généralement très complexes à cause du fait que ces paramètres ne sont pas indépendants les uns des autres. Il s'avère donc primordial de décrire le choix de ces derniers et de leurs implantations dans les systèmes existants.

2.1 Choix du design

Pour concevoir une DSM, il faut trancher sur les points suivants: la structure et la granularité des données partagées, leurs accès, leur cohérence et les problèmes liés à l'hétérogénéité du système [Nitzberg 91, Balter 91]

2.1.1 Structure et granularité

Lorsqu'on parle de structure, on fait référence à l'unité sémantique des données partagées en mémoire. La granularité du partage définit la grandeur de l'unité. Il peut s'agir d'octets, de mots, de pages ou de structures de données complexes.

Parmi les systèmes qui ont été mis au point, on en trouve qui ne structurent pas leur mémoire. Nous pouvons citer IVY et DASH, dont les unités de partage sont respectivement des pages de 1 kiloctets et de 16 octets [Nitzberg 91, Lenoski 90]. D'autres, par contre, structurent leurs mémoires. LINDA organise sa mémoire partagée comme une base de données [Carriero 89] et MUNIN la structure en objets [Bennett 90]. Cependant, il faut noter que:

- Une grande granularité augmente le débit du système car le nombre de messages circulant dans le système est réduit. Par contre, la probabilité que deux processus accèdent à la même unité de partage est grande.
- Une petite granularité diminue le débit du système car le nombre de messages circulant dans le système est élevé. Par contre, la probabilité que deux processus accèdent à la même unité de partage est faible.

Au niveau des serveurs, il serait plus avantageux d'avoir une grande granularité, ce qui réduirait la taille des structures nécessaires pour identifier les données et accélérerait leurs accès. Par contre au niveau des machines autonomes, on gagnerait à avoir une granularité plus petite pour permettre l'utilisation des structures de données fournies par les langages de programmation procéduraux tels que Pascal, C, etc.

2.1.2 La sémantique de la cohérence

Le programmeur doit comprendre comment la mise à jour des données partagées se réalise à travers le système. Un système implante l'un des types de cohérences suivants: stricte, séquentielle, processeur, faible ou libre [Nitzberg 91].

Pour une cohérence stricte, une lecture retourne toujours la valeur la plus récemment écrite. Quelques exemples de systèmes mis au point sont MERMAID, MIRAGE, IVY et SHIVA [Nitzberg 91].

Dans une cohérence séquentielle, l'exécution d'un programme réparti donne exactement le même résultat si l'exécution s'était effectuée sur un seul processeur. L'ordre de l'exécution des processeurs doit suivre l'ordre imposé par le programme.

Pour une cohérence processeur, les requêtes d'écriture générées par un processeur sont toujours exécutées en ordre. Cependant les écritures générées par des processeurs différents peuvent être exécutées en désordre. Cette cohérence est implantée dans PLUS [Bisiani 90].

Dans le cas d'une cohérence faible, une cohérence séquentielle est imposée pour l'accès aux variables de synchronisation, par contre il n'y a pas de contrainte quant à l'accès aux autres types de variables. Elle est utilisée dans DASH [Lenoski 90].

Pour une cohérence libre, il s'agit d'utiliser une consistance faible avec deux opérations de synchronisation: acquérir et libérer; chaque type d'opération doit être implanté selon une

cohérence de type processeur. DASH l'utilise dans son système.

La gestion de la cohérence peut être transparente à l'usager ou effectuée manuellement par le programmeur. Plusieurs programmeurs prennent pour acquis que dans les systèmes à DSM, le résultat d'une séquence d'écritures générée par un processeur est directement visible par les autres processeurs dans le même ordre. Ces systèmes sont cohérents séquentiellement et laissent au programmeur le choix d'utiliser des opérations de lecture et d'écriture pour implanter les primitives de synchronisation. Lorsqu'il y a plusieurs mémoires physiques, la mise au point d'une cohérence stricte ou séquentielle est très coûteuse en terme de temps, car au niveau du système, toutes les opérations sont traitées en série. Cependant, dans plusieurs applications, des formes de cohérences dans lesquelles les actions générées par un poste ne sont pas immédiatement détectables par les autres postes sont suffisantes. Typiquement, un programme parallèle alterne entre plusieurs opérations de lecture successives et des écritures sur des variables partagées ou des variables de synchronisation. Ainsi, imposer un ordre strict sur toutes des opérations de lecture et d'écriture n'est pas nécessaire si le programmeur comprend que les opérations de synchronisation doivent être utilisées dans certain cas spécifiques seulement; par exemple lorsqu'une opération de lecture suit immédiatement une opération d'écriture sur la même donnée. Dans ces cas, pour que les programmes donnent des résultats corrects, le programmeur doit explicitement utiliser des opérations sur des variables de synchronisation. Il faut noter que plus on avance vers les formes de cohérences plus faibles, un gain de performance est fait au coût d'un modèle de programmation plus complexe et difficile à comprendre par l'usager. Ce gain en performance est dû au fait que le fonctionnement du système requiert moins de synchronisation.

2.1.3 Hétérogénéité

Il s'agit de greffer une mémoire partagée sur des machines ayant des architectures différentes. Ceci parait difficile car ces machines peuvent utiliser des représentations différentes pour leurs types de données de base tels que les entiers et les réels. Il est plus facile d'implanter une DSM si la mémoire est structurée en variables ou en objets dans un langage de haut niveau. Ainsi il devient possible à un compilateur DSM d'insérer des routines de conversion pour accéder à la mémoire partagée. AGORA utilise cette approche [Nitzberg 91]. MERMAID pour sa part utilise une nouvelle approche. Sa mémoire est partagée en pages et une page ne peut contenir qu'un seul type de données. Dès qu'une page passe par deux architectures différentes, une routine convertit ses données dans un format approprié [Nitzberg 91].

Il est possible d'avoir des DSM hétérogènes. Cependant, le temps mis pour faire la conversion des formats peut affecter les performances du système. Puisqu'un des buts visés lors de la conception des DSM est d'obtenir des performances élevées, il est préférable d'utiliser des machines homogènes.

2.1.4 Extensibilité

Les systèmes à DSM doivent supporter le plus grand nombre possible de processeurs. Pour le faire, il faut éviter des situations pouvant donner naissance aux goulots d'étranglement. On a donc intérêt à:

- privilégier des traitements répartis au détriment des traitements centralisés;

- favoriser les solutions ne donnant pas de goulots d'étranglement;
- éviter le plus possible des opérations qui font intervenir tous les postes du système comme la diffusion car ils surchargent le réseau de communication.

Il faut aussi noter que le choix de la topologie du réseau de communication est très déterminant quant à l'agrandissement du système. Un exemple de système est IVY qui utilise ETHERNET et qui est limité à 100 noeuds alors que SHIVA qui utilise l'hypercube iPSC/2 d'INTEL peut supporter plus de postes [Nitzberg 91].

2.2 Implantation

Une DSM devrait transformer un accès à une variable partagée en une communication interprocessus. Ceci requiert des algorithmes pour localiser les données partagées, les maintenir cohérentes et les mettre à jour.

2.2.1 Accès et localisation des données

Dans un système à DSM, tout programme parallèle doit être capable de localiser et d'accéder aux données dont il a besoin. On distingue une distribution statique où les données sont réparties dans chaque site selon un algorithme de répartition. Selon ce schéma, un site peut devenir surchargé, et même maintenir des variables qu'il n'utilise plus.

La distribution dynamique répartit les données là où elles ont été utilisées. Dans ce cas la localisation d'une donnée devient difficile car la donnée change de site au fur et à mesure qu'elle est utilisée. La solution naturelle est l'utilisation d'un serveur centralisé. Le serveur peut poser des problèmes car les requêtes qui lui sont soumises sont traitées séquentiellement. Il peut être surchargé car il peut devenir très sollicité ce qui augmente le temps d'accès aux variables partagées. Tous ces facteurs contribuent à réduire l'efficacité du traitement parallèle. Une stratégie possible consiste à décentraliser le serveur. Dans ce cas, chaque requête se traduit par une diffusion à travers le réseau et seul le site qui possède cette donnée répond à la requête. S'il y a un grand partage entre les données, le réseau devient rapidement surchargé, ce qui a pour conséquence de réduire l'efficacité du système.

Une autre solution consiste à utiliser le schéma de la répartition du propriétaire des données partagées [Nitzberg 91]. Chaque donnée partagée est associée à un propriétaire qui est en fait le site privilégié et ayant la copie de la donnée. Cependant, au fur et à mesure que les données transitent à travers le réseau, le propriétaire change également. Si un site veut une copie de la donnée, il envoie une requête au propriétaire reconnu de cette donnée. Si celui-ci possède toujours la dernière copie mise à jour, il la lui envoie. Par contre si le propriétaire a transféré cette donnée à un autre, il transmet la requête au nouveau propriétaire. Le seul problème avec ce schéma est la surcharge éventuelle du réseau car pour trouver la donnée, il faut consulter tous les sites par où elle a transité. Ceci augmente le temps de réponse de la requête. La solution est de maintenir une liste de tous les propriétaires probables. C'est la technique réalisée par IVY [Nitzberg 91]. Ainsi, dans chaque site et pour chaque donnée partagée, on dispose d'une liste de tous les propriétaires potentiels qu'il suffit d'aller consulter pour avoir la localisation de la donnée voulue.

Lorsqu'il y a réplication de données, une DSM devrait pouvoir localiser toutes les copies existantes dans le système. Deux grandes familles de méthodes sont utilisées. Il s'agit des techniques à répertoire et à chaînage.

Dans les systèmes DASH [Lenoski 90], la réunion des mémoires locales de chaque site constitue la mémoire globale. Ainsi, cette mémoire globale n'est formée que d'une mémoire logique et pour laquelle chaque portion de cette mémoire est gérée par le site où se trouve la portion. Les sites emploient un répertoire réparti, implanté matériellement. Pour un site donné, le répertoire contient l'adresse de tous les blocs physiques qui se trouvent dans sa portion de la mémoire globale. Chaque bloc est représenté par un répertoire qui spécifie son état et où trouver toutes les autres copies. Le site où est disponible ce répertoire est appelé le *site initial*. Ainsi, pour localiser une donnée ne se trouvant pas dans un poste, il faut communiquer avec le *site initial*.

IVY [Nitzberg 91] pour sa part, utilise à chaque site, une table des pages qui contient pour chaque page, la liste des sites probables possédant la page. Ainsi, pour localiser une donnée, il faut consulter les propriétaires probables.

Par contre, PLUS [Bisiani 90] utilise une liste chaînée répartie pour maintenir une trace des copies répliquées. Cette liste a deux champs d'information. Le premier est l'identité du site ayant la première copie à consulter, et le second est l'identité du prochain site ayant une copie. Chaque site possède une table de pages qui associe le numéro d'une page au site le plus proche possédant une copie. Ainsi, pour accéder à toutes les copies d'une donnée, il faut consulter la table des pages et la liste des sites voisins afin d'obtenir l'adresse du site le plus proche ayant une copie de cette donnée. Par la suite il faut

consulter la liste des copies disponibles sur le site obtenu précédemment.

2.2.2 Protocole de cohérence

S'il n'existe qu'une seule copie d'une donnée partagée à travers le système, tous les accès à cette donnée sont traités séquentiellement car le site qui a la donnée reçoit des requêtes et les traite au fur et à mesure qu'elles arrivent. Pour accroître le parallélisme, il faudrait duplifier les données; ainsi plusieurs accès à la même donnée peuvent se faire simultanément. On distingue deux types de protocoles de cohérence: l'invalidation en écriture et la mise à jour en écriture.

2.2.3 Protocole d'invalidation en écriture

Ce protocole partage les copies en deux groupes d'usagers: ceux qui peuvent accéder à la donnée en lecture et ceux qui peuvent la modifier. Ici, il peut y avoir plusieurs copies d'une donnée que l'on peut accéder en lecture uniquement et une seule copie de la donnée que l'on peut accéder en écriture. Le protocole invalide toutes les autres copies sauf celle qui peut être accédée en écriture.

Lors d'un accès en lecture, la donnée est obtenue localement si elle est valide. Autrement, une demande de lecture est transmise à un site possédant une copie valide et cette copie est retournée. La copie demeure valide jusqu'à ce qu'une demande d'invalidation soit reçue.

Pour un accès en écriture, la modification de la donnée peut être satisfaite immédiatement si cette donnée n'est pas partagée avec un autre site et que le site auquel elle réside a le privilège d'écriture. Par contre si la donnée est partagée et que le site possède le privilège d'écriture, alors le poste envoie une requête d'invalidation à tous les sites ayant une copie valide. Dès que l'opération d'invalidation est complétée, la donnée peut être modifiée.

Exemple: protocole d'invalidation en écriture de DASH

DASH est constitué de plusieurs sites reliés par un réseau de communication à haut débit. La mémoire globale est répartie sur chaque site et est accessible par tous. Chaque site est formé de plusieurs processeurs ayant chacun une antémémoire, une portion de la mémoire partagée, une antémémoire partagée pour les accès à distance, un répertoire pour les blocs qui se trouvent dans la portion de la mémoire globale du site, ainsi qu'une interface de communication. Pour présenter le protocole de cohérence, nous utilisons les désignations suivantes: Le *processeur local* est le processeur qui génère la requête; Les *processeurs voisins* sont les processeurs qui partagent la même mémoire locale; le *site initial* est le site qui dispose du répertoire renfermant les informations pour localiser la donnée. Un *site distant* est n'importe quel autre site; la *mémoire locale* fait référence à la portion de la mémoire globale associée au *processeur local* alors que la *mémoire distante* est n'importe quelle autre mémoire.

Un bloc en mémoire peut être dans l'un des trois états suivants: PRIVÉ, s'il n'est présent dans aucun autre poste; PARTAGÉ, s'il est présent dans au moins un autre poste et qu'il n'est pas modifié; SALE, s'il est présent dans un seul poste et qu'il est modifié. Le protocole définit la notion de site propriétaire pour chaque bloc mémoire. Le site

propriétaire est le *site initial*. Cependant, dans le cas où le bloc mémoire est présent dans un poste mais dans l'état SALE, alors celui-ci devient le propriétaire. Seul le propriétaire de la donnée peut autoriser une modification de la donnée et mettre à jour l'état du répertoire.

Comme pour les blocs mémoires, un bloc dans une antémémoire peut être dans l'un des états suivants: INVALIDE, PARTAGÉ, ou SALE. Dans l'état PARTAGÉ, il y a plusieurs processeurs qui partagent le bloc. L'état SALE signifie qu'il y a un seul processeur qui contient une copie du bloc mémoire et que cette copie a été modifiée.

Les sections suivantes décrivent les primitives que le protocole de cohérence de DASH met à la disposition du système: *la lecture*, l'*écriture*, et l'*écriture en mémoire*.

La primitive *lecture*

Lors d'une lecture, si le bloc est présent dans l'antémémoire du poste qui la demande, alors l'antémémoire s'occupe de transférer la donnée au *processeur local*. Par contre, si le bloc est absent, alors le demandeur communique avec les postes avec lesquels il partage la même portion de la mémoire globale. Si le bloc est présent dans l'une des antémémoires et qu'il est à l'état PARTAGÉ, alors la donnée est tout simplement transférée à travers le bus jusqu'au demandeur et il n'y a pas d'accès au *site initial*. Si par contre l'état du bloc est SALE, alors ce site va récrire le bloc en mémoire au *site initial* et transférer la donnée au demandeur. Si la lecture ne peut être satisfaite localement, alors le demandeur transfère sa requête au *site initial* qui obtient l'état de ce bloc en mémoire. Si le bloc est PRIVÉ ou PARTAGÉ, alors la donnée est transférée au demandeur par le contrôleur du répertoire. Il

enregistre aussi le fait que le demandeur a une copie de la donnée. Si le bloc par contre est dans l'état SALE, alors la requête est transférée au site où se trouve la copie SALE.

Ce site propriétaire envoie deux messages en réponse à la demande de lecture. Un message contenant la donnée est envoyée directement au demandeur et une demande de mise à jour est envoyée au site maison. Cette demande d'écriture écrit le contenu du bloc SALE en mémoire et en même temps met à jour le répertoire.

La primitive *écriture*

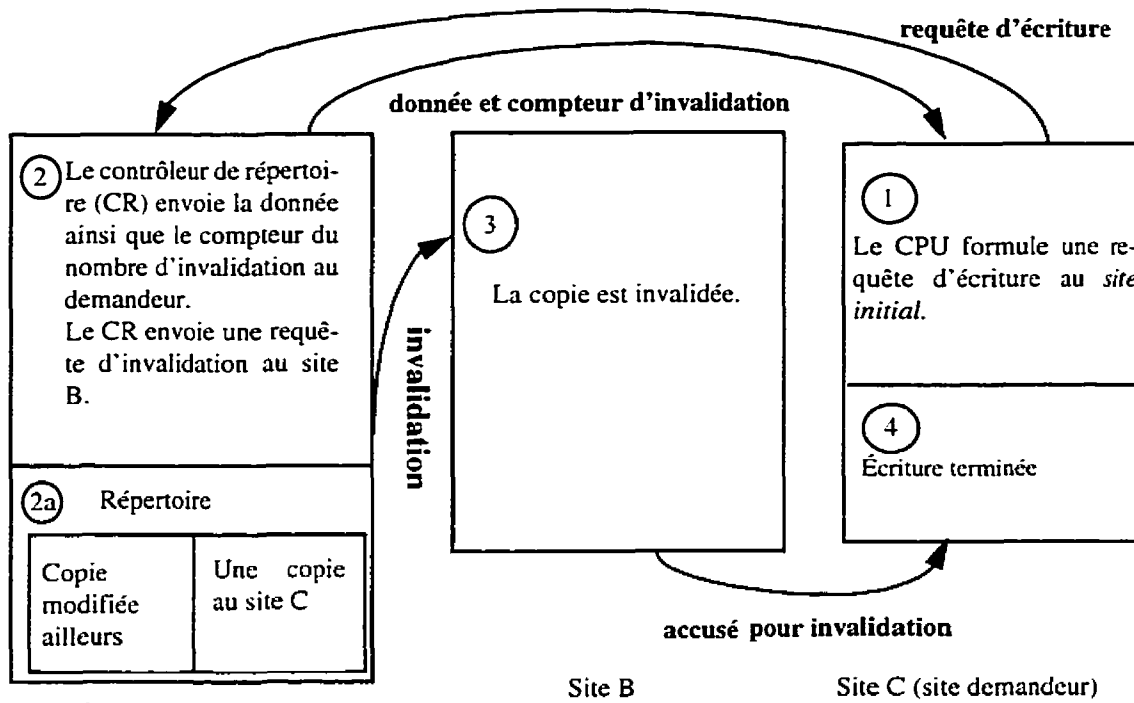
Lors d'une écriture, si la donnée ne peut être satisfaite par le demandeur, alors elle est transmise aux *processeurs voisins* à travers le bus. Si dans l'antémémoire d'un des processeurs on trouve le bloc et que celui-ci est à l'état SALE, alors ce poste transfère le bloc et invalide sa propre copie. Si la donnée ne se trouve pas dans les postes locaux, alors la requête est transférée au site maison. Si le bloc mémoire est PRIVÉ ou PARTAGÉ, alors la donnée est transférée au demandeur ainsi qu'un message indiquant qu'il est le nouveau propriétaire. En plus, si la donnée est PARTAGÉE, alors un message d'invalidation est envoyé à chaque poste contenant le bloc. La donnée ainsi qu'un compteur du nombre d'invalidations sont également envoyées au demandeur. Chacun de ces postes après avoir complété son invalidation renvoie un accusé de réception au demandeur. Si le répertoire indique que le bloc est SALE, alors la requête est transférée au poste ayant cette copie. Ce poste invalide sa copie et envoie un message au demandeur, comportant la donnée et le fait qu'il est le nouveau propriétaire du bloc. Une requête est envoyée également au *site initial* afin de mettre à jour le propriétaire du bloc. Après avoir reçu le message, le *site initial* en informe le nouveau propriétaire.

La figure 2.1a illustre une opération d'écriture dans le cas où la donnée est partagée. Le site demandeur est le site C. Le site A est le *site initial* alors que le site B contient une copie du bloc. Le site demandeur formule sa requête et l'envoie au *site initial*. Celui-ci après avoir reçu la requête met à jour son répertoire de copies et envoie un message d'invalidation au site B ayant une copie valide de la donnée. Il envoie également au demandeur un compteur pour le nombre d'accuser de réception à recevoir pour mettre fin à l'opération d'écriture. Dès que le site B finit d'invalider sa copie, il envoie un accusé d'invalidation au demandeur. La réception de cet accusé par le demandeur permet de mettre fin à cette transaction.

La figure 2.1b illustre une opération d'écriture dans le cas où la donnée est SALE. Le site B est le site demandeur, le site A le *site initial* et le site C le site propriétaire de la donnée. Le demandeur génère sa requête et la transmet au *site initial*. Celui-ci envoie la requête au site propriétaire de la donnée. Ce dernier transmet le bloc au demandeur, et envoie l'identité du nouveau propriétaire au *site initial*. Le *site initial* met à jour son répertoire de copies et envoie un accusé au demandeur. La réception de cet accusé par le demandeur met fin à l'opération d'écriture.

La primitive *écriture mémoire*

Tout bloc SALE contenu dans une antémémoire et qui doit être remplacé doit être récrit en mémoire locale. Si le bloc est dans le *site initial*, alors le bloc est tout simplement récrit en mémoire principale. Si par contre il ne se trouve pas dans le *site initial*, alors un message est envoyé au *site initial* qui met à jour la mémoire principale et change l'état du bloc en PRIVÉ.



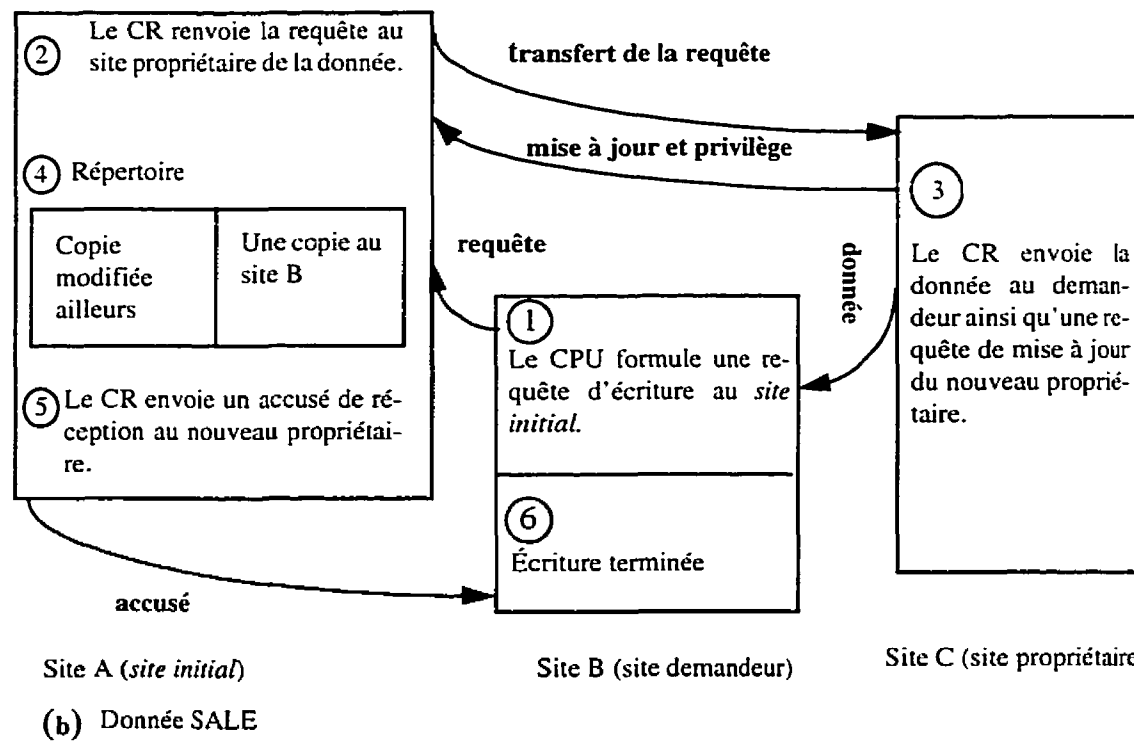


Figure 2.1 Protocole d'invalidation en écriture de DASH

2.2.4 Protocole de mise à jour en écriture

Ce protocole partage les copies en deux groupes d'usagers: ceux qui peuvent accéder à la donnée en lecture et ceux qui peuvent la modifier. Le protocole met à jour toutes les autres copies du système lors d'une opération d'écriture.

Lors d'un accès en lecture, la donnée est obtenu localement si elle est valide. Autrement, une demande de lecture est transmise à un site possédant une copie valide et cette copie est retournée. La copie demeure intacte jusqu'à ce qu'une demande de mise à jour soit reçue. Pour un accès en écriture, la modification de la donnée peut être satisfaite immédiatement si cette donnée n'est pas partagée et que le site sur lequel elle réside a le privilège d'écriture. Par contre si la donnée est partagée avec un privilège d'écriture, alors le poste envoie une requête pour obtenir le privilège d'écriture ainsi qu'une requête pour une copie de la donnée si la copie locale n'est pas valide. Dès que la donnée devient disponible au site, il la modifie et demande une mise à jour des autres copies du système.

Exemple: protocole de mise à jour en écriture de PLUS [Bisiani 90]

Pour localiser et accéder aux données répliquées, PLUS utilise le protocole de mise à jour en écriture. PLUS est constitué de plusieurs postes reliés par un réseau de communication à haut débit. Chaque site est comprend un processeur muni de son antémémoire, une mémoire locale et un gestionnaire de la cohérence mémoire (GCM) qui fait le lien entre la mémoire et le réseau. La mémoire locale est utilisée comme une mémoire vive et comme une antémémoire pour les données qui sont absentes localement et qui se trouvent dans les mémoires des sites.

Une table des pages réunit, la liste des pages physiques répliquées dans les différents sites. Le premier élément de cette liste est l'identité du site ayant la *copie maîtresse* c'est-à-dire la première copie créée. L'adresse globale d'une page physique est le couple formée par l'identité du site ainsi que le numéro de la page que nous notons << *site.id*, *page.id*>> qui est généré directement par le mécanisme de correspondance de la mémoire du processeur. Le champ *site.id* de l'adresse physique détermine quel site est adressé et le champ *page.id* spécifie le numéro de la page dans ce site. Dans chaque site, l'identité des sites ayant une copie répliquée est accessible au GCM, par la liste des copies qui comporte deux champs dont un pour l'adresse de la copie maîtresse et l'autre pour l'adresse de la prochaine copie. Les sections suivantes décrivent les primitives de lecture et d'écriture du protocole de cohérence de PLUS.

La primitive *lecture*

Si le site local est indiqué, alors la mémoire locale est lue. Autrement, le GCM envoie une requête de lecture au site identifié, attend la réponse et envoie la donnée au processeur.

La primitive *écriture*

Cette opération est plus compliquée car elle va porter sur toutes les copies. L'écriture est d'abord effectuée sur la copie maîtresse avant d'être propagée sur les autres copies contenues dans la table des pages.

Si l'adresse physique indique un autre site, alors le GCM envoie une requête d'écriture à ce site. Autrement, il consulte la table de la copie maîtresse pour localiser la première

copie. Si la copie maîtresse est locale, alors il procède à l'écriture de la mémoire locale et envoie un message de mise à jour à la prochaine copie si jamais il y en a une. Si par contre la copie maîtresse n'est pas locale, alors la requête est transférée au site contenant la copie maîtresse.

Un GCM qui reçoit une requête d'écriture de son site demande une mise à jour par la copie maîtresse puis l'opération se propage sur les prochaines copies. Finalement, le site contenant la dernière copie sur la liste des copies envoie un accusé de réception au processeur qui a généré la requête originale. Ainsi se termine la requête.

La figure 2.2 représente le protocole mise à jour en écriture de PLUS. Il s'agit d'une opération d'écriture. Tous les sites ont une copie de la donnée sauf le site D qui est demandeur. Le site A est le site ayant la copie maîtresse. La requête d'écriture est formulée par le demandeur puis envoyé au site B. Celui-ci consulte sa liste des copies pour obtenir l'adresse du site ayant la copie maîtresse et lui transmet la requête. Celui-ci met à jour sa copie et transmet la requête au site B ayant la prochaine copie. Le site B procède de la même façon. Dès que la requête parvient au site C, la liste des copies est terminée et ce dernier envoie un accusé au demandeur. On constate que les mises à jour commencent toujours par la copie maîtresse et ensuite se propagent sur les autres sites de la liste des copies. L'opération d'écriture est terminée lorsque le dernier noeud de la liste envoie un accusé de réception au demandeur de la requête d'écriture.

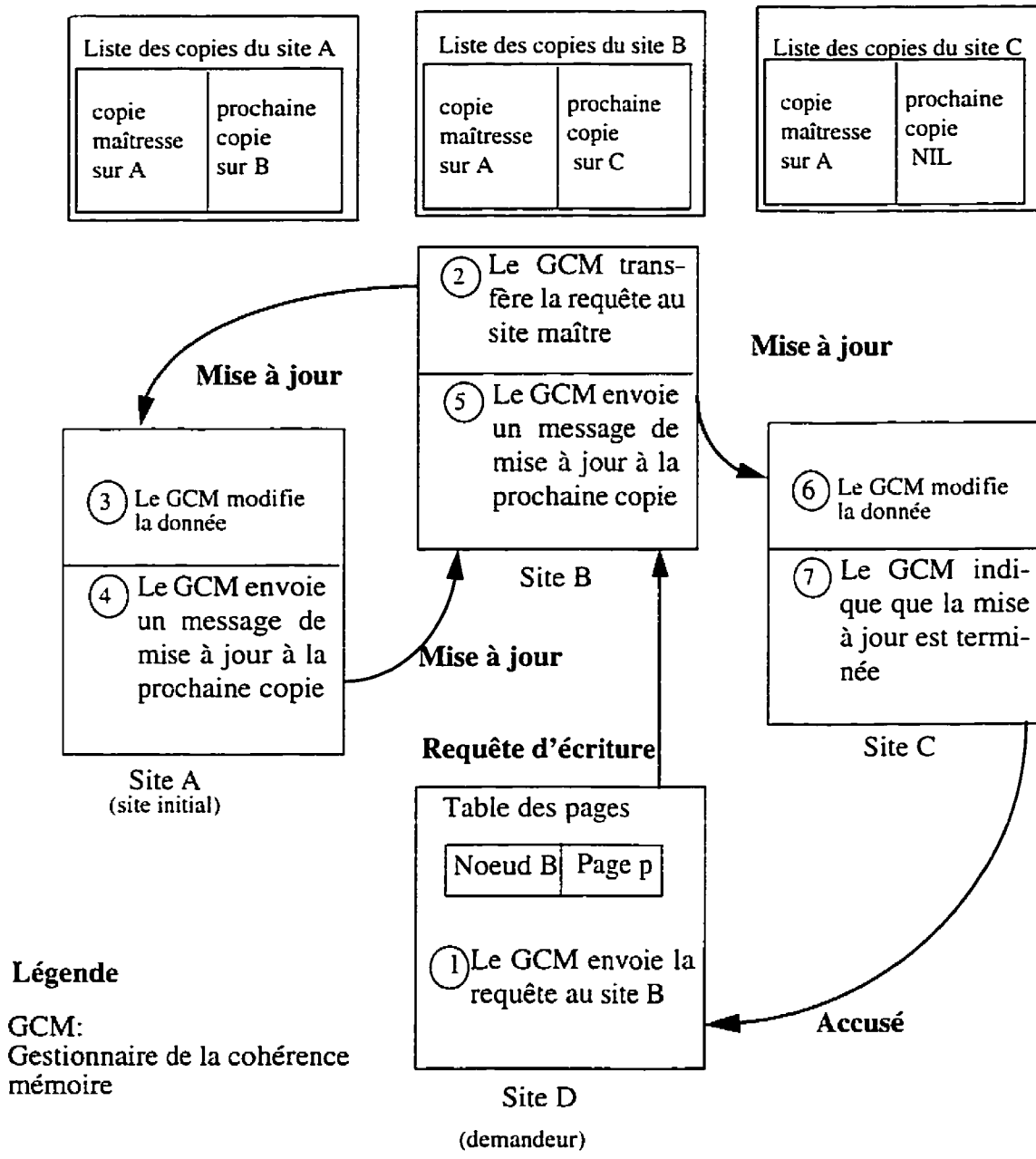


Figure 2.2 Protocole mise à jour en écriture de PLUS

2.2.5 Stratégies de remplacement

À la suite d'une faute d'objet [Tanenbaum 87], le GCM doit, pour libérer de la place mémoire, retirer un objet de sa mémoire pour l'objet manquant. Si l'objet a été modifié depuis son chargement en mémoire, il faut l'écrire en zone sûre sinon l'objet lu remplace simplement l'objet supprimé.

Le choix de l'objet peut se faire au hasard. Mais si on choisit un objet qui sera peu utilisé, on améliore sensiblement les performances du système. Si par contre on choisit un objet très demandé, alors il est possible que des références ultérieures lui soient adressées ce qui entraîne une perte de temps inutile. Plusieurs algorithmes de remplacement ont été décrits dans la littérature. Nous allons en présenter dans les paragraphes suivants quelques algorithmes parmi les plus utiles.

1 - Algorithme optimal

Lors d'une faute d'objet, l'algorithme optimal choisit un objet à remplacer qui ne fera pas l'objet d'une référence ultérieure, ou à défaut, un objet auquel on fera la référence la plus tardive. L'algorithme optimal suppose la connaissance à priori de l'ensemble des chaînes de référence, c'est-à-dire l'identité des objets auxquels on accédera.

2- Remplacement d'un objet non récemment utilisé.

Dans cet algorithme, deux bits d'information sont associés à chaque objet utilisé. Le premier bit R ou bit de référence est positionné par le matériel à chaque lecture ou écriture

de l'objet. Le second bit M ou bit de modification est positionné quand on écrit dans l'objet. Ces bits doivent être mis à jour à chaque référence. Il est donc important qu'il soit fait par le matériel. Dès qu'un bit est mis à 1, il reste dans cet état jusqu'à ce que le GCM vienne le mettre à 0. Le fonctionnement du système est le suivant: au déclenchement d'un processus (programme en exécution), le GCM met à 0 les bits de tous les objets. Périodiquement (par exemple à chaque interruption d'horloge), le bit R est mis à 0 pour différencier les objets qui n'ont pas été récemment référencés des autres. Lorsqu'une faute de d'objet se produit et que la mémoire est pleine, le GCM parcourt tous les objets et les répartit dans l'une des quatre catégories suivantes en fonction des différentes valeurs des bits R et M:

0: non référencé, non modifié;

1: non référencé, modifié;

2: référencé, non modifié;

3: référencé, modifié.

Ces différentes catégories permettent plus tard de faire un choix de l'objet à remplacer.

Il faut noter que les interruptions ne réinitialisent pas le bit M car il permet de savoir si l'objet a été modifié ou non. L'algorithme de l'objet le moins récemment utilisé retire un objet au hasard de la catégorie qui à le plus petit numéro. Cet algorithme est relativement efficace et fournit des performances qui sont souvent suffisantes même si elles ne sont pas optimales.

3- Cercle chronologique d'utilisation (Least Recently Used (LRU))

Puisque les objets récemment utilisés ont une probabilité plus élevée d'être utilisés dans un futur proche, une objet inutilisé depuis longtemps a une probabilité faible d'être réutilisé prochainement. On choisit donc comme victime l'objet ayant fait la référence la plus ancienne. Cet algorithme est théoriquement réalisable mais très coûteux. En effet, il faut mémoriser une liste chaînée de tous les objets en mémoire, l'objet le moins utilisé étant à la fin de la liste et le plus utilisé à la tête de la liste. La difficulté de cet algorithme vient du fait que cette liste doit être mise à jour chaque fois que la mémoire est adressée. Localiser un objet, le supprimer puis la déplacer au début de la liste sont des opérations très coûteuses en temps de processeur.

4- Cercle chronologique de changement (FIFO)

Il s'agit d'une approximation de la précédente. Cet algorithme choisit comme victime l'objet le plus anciennement chargé. Pour le faire, le GCM maintient une liste de tous les objets en mémoire, l'objet le plus ancien étant en tête et l'objet le plus récent étant à la fin. Dès qu'il faut remplacer un objet, la premier objet de la liste est retiré et on insère le nouvel objet à la fin de la liste.

Une modification simple de l'algorithme FIFO consiste à examiner le bit R et M de l'objet le plus ancien. Si cet objet appartient à la catégorie 0 (non référencé et mon modifié), cet objet est supprimé. Sinon on vérifie le bit R et M de l'objet un peu moins ancien et ainsi de suite. S'il n'y a pas d'objet de la catégorie 0, on applique la même procédure aux catégories 1, 2, puis 3.

5- Algorithme de la seconde chance

C'est une variante de l'algorithme précédent. L'idée ici est de tester le bit R de l'objet le plus ancien. S'il est à 0, l'objet est immédiatement remplacé. Sinon ce bit est mis à 0 et l'objet est mis en fin de la liste des objets comme s'il venait d'être chargé en mémoire et la recherche continue. Cet algorithme cherche donc un ancien objet qui n'a pas été référencé. Si tous les objets ont été référencés, l'algorithme de la seconde chance est équivalent à l'algorithme FIFO parce que la liste sera ordonnée de manière à avoir l'objet le plus anciennement référencé à la tête et le plus récemment référencé à la fin.

2.3 Analyse de performance des systèmes à DSM

Dans la section précédente, nous avons passé en revue les différents paramètres et algorithmes à tenir compte pour l'implantation d'une DSM. Cependant pour avoir un estimé des performances attendues du système une fois construit. il faut recourir à l'analyse de performance.

Des évaluations de performance des protocoles de cohérence des multiprocesseurs ont déjà été réalisées. Un modèle de simulation a été proposé par Achibald et Baer [Archibal 86] pour comparer les différents protocoles. Une technique basée sur les réseaux de Pétri temporisés généralisés a été décrite dans [Vernon 86], mais la complexité de la technique la rend peu appropriée pour les systèmes de grande envergure. Le modèle des réseaux de file d'attente a d'abord été utilisé pour la modélisation des réseaux à commutation de paquets [Yang 88]. Par la suite ces concepts furent appliqués à l'analyse de performance des multiprocesseurs. Qing Yang et Laxmi N. Bhuyan proposent un concept basé sur la

décomposition hiérarchique [Bhuyan 89]. Ils utilisent un modèle à deux niveaux pour obtenir les mesures de paramètres caractéristiques du système. Le modèle de haut niveau est constitué d'une chaîne de Markov dont les états représentent les différents états que peuvent prendre les données dans les antémémoires du système dû au protocole de cohérence. Au bas niveau, les taux des transitions entre les états de la chaîne de Markov du niveau supérieur peuvent être calculés en résolvant le réseau de file d'attente qui représente le système physique et logique. Le modèle de haut niveau fournit des probabilités stationnaires pour la détermination des probabilités de routage dans le modèle de bas niveau. La solution complète est obtenue en itérant d'un modèle à l'autre jusqu'à convergence.

Ce chapitre nous a permis de constater que la conception des systèmes DSM est très complexe car il faut faire des choix qui ne sont pas nécessairement indépendants et qui sont d'autant plus importants qu'ils affectent directement la performance du système. Le chapitre suivant sera dont consacré à la spécification physique du système et de son fonctionnement logique.

CHAPITRE 3

LES MODÈLES

Dans le chapitre précédent, nous avons présenté quelques paramètres nécessaires pour la conception d'une mémoire partagée répartie ainsi que décrit leurs influences sur les performances du système qui en résulte. Nous avons également décrit la méthodologie utilisée pour évaluer les performances de ces systèmes. Il ressortait de cette description que la spécification du système physique et de son protocole de cohérence étaient suffisant pour sa modélisation analytique. Dans le présent chapitre, nous allons nous inspirer de cette étude pour décrire l'architecture de notre système ainsi que son fonctionnement logique.

3.1 Le modèle physique

Le système que nous voulons analyser peut être représenté par le modèle de la figure 3.1.

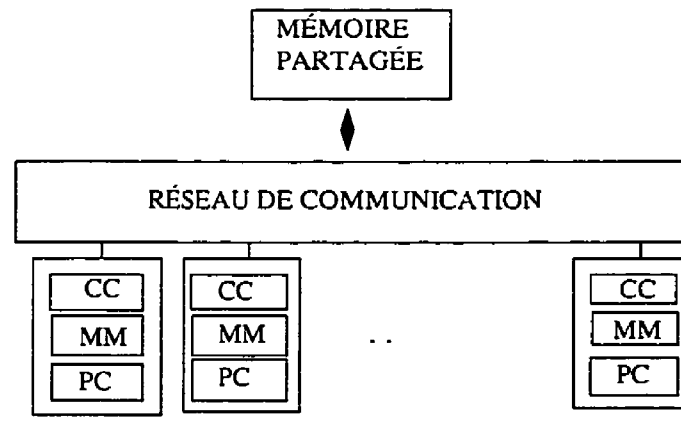


Figure 3.1 Le système physique

Dans ce modèle, plusieurs postes de travail sont reliés par un réseau à grande vitesse. Chaque poste de travail possède un processeur de calcul (PC), un module mémoire (MM) et un ou plusieurs coprocesseurs de communication (CC). Le but de notre travail est d'implanter une mémoire partagée à partir des modules mémoires disponibles localement à chaque poste de travail.

Le réseau de communication fournit une interface au niveau transport sur laquelle nous faisons les hypothèses ci-dessous.

- Tous les postes peuvent communiquer entre eux en émettant des messages. Il n'y a donc pas de poste isolé du reste du réseau.
- Il n'y a pas d'altération, ni de duplification, ni de perte des messages émis par les postes, c'est-à-dire que le délai d'acheminement des messages est non nul mais fini.
- Il n'y a pas de déséquencement des messages, c'est-à-dire que les messages sont reçus par un site, dans le même ordre qu'ils ont été émis par le site émetteur.
- Un site peut détecter une panne du système de communication ou d'un autre poste. Dans le cas où c'est le réseau qui est défectueux il annule sa requête. Dans le cas où c'est le poste récepteur qui est défectueux, il change de destinataire.

Le système est asynchrone en ce sens qu'il n'y a pas une horloge centrale qui diffuse son heure aux autres sites. Lorsqu'un poste a une donnée à transmettre, il formule une requête en spécifiant son adresse et celle du récepteur dans un paquet et le transmet au réseau. Une fois le paquet transmis, le réseau redevient libre et peut être utilisé par un autre poste. Le récepteur envoie une réponse de manière similaire à l'émetteur.

Pour implanter cette mémoire partagée, nous allons utiliser une approche serveur. Cependant, l'utilisation d'un seul serveur de mémoire n'est pas très indiqué car non seulement il peut devenir surchargé, mais aussi en cas de panne de ce dernier, tout le système est paralysé. Pour prévenir les goulots d'étranglement éventuels et augmenter la tolérance aux pannes du système, il faut utiliser plusieurs serveurs. Le rôle des serveurs est double: ils doivent non seulement répondre aux requêtes issues des postes de travail mais aussi servir de lieu de sauvegarde d'information afin d'assurer une reprise en cas de panne. Si nous partageons l'ensemble des serveurs en deux grands groupes et leur imposons les tâches précédentes, alors nous venons de créer une certaine hiérarchie entre eux. Ainsi le premier groupe sera celui des serveurs actifs que nous qualifierons de *serveurs principaux* (SP). Ils seront chargés de répondre aux requêtes issues des postes de travail (ou encore de gérer le trafic prioritaire) et de générer les requêtes pour la sauvegarde des données. Le second groupe est celui des serveurs passifs que nous qualifierons de *serveurs secondaires* (SS) dont le rôle est de répondre aux requêtes en provenance des SP et d'assurer une reprise en cas de panne de ces derniers.

Ainsi le système final est celui de la figure 3.2. Il s'agit du système de la figure 3.1 dans lequel nous assignons à M postes de travail le rôle de SP et à L autres, celui de SS. L'utilisation de plusieurs serveurs est donc bénéfique pour le système mais le problème posé reste celui de leur nombre et de leur gestion.

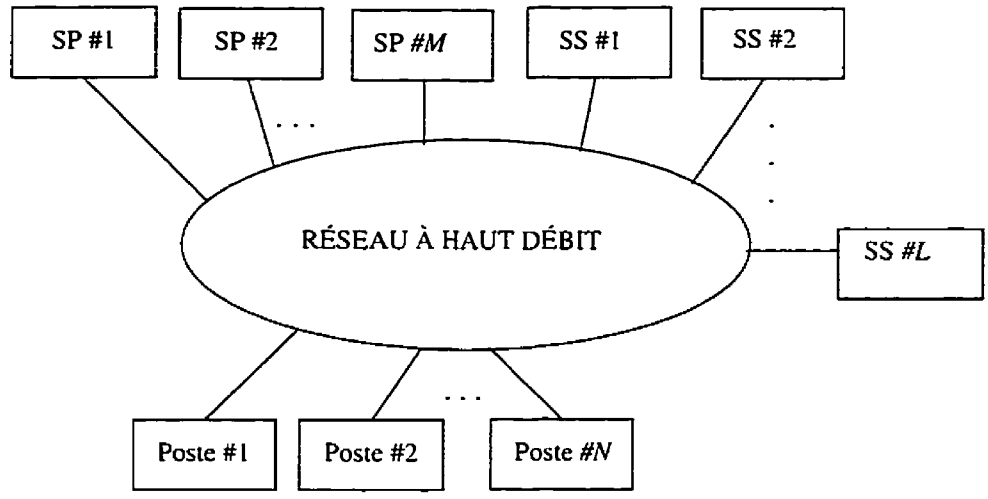


Figure 3.2 Modèle à analyser

Le système global peut être divisé en deux sous-systèmes ayant des fonctions différentes. Le premier sous-système est constitué de postes de travail, du réseau et des serveurs principaux. C'est dans cette partie du système que les programmes parallèles s'exécutent. Le second sous-système est constitué des serveurs et du réseau. C'est là que résident les données partagées. Lors du fonctionnement, les serveurs principaux ont la responsabilité de mettre à jour les serveurs secondaires. Les informations à mettre à jour sont les données du programme ainsi que les informations sur leurs localisations pour assurer une reprise en cas de panne. En cas de surcharge d'un serveur principal, un serveur secondaire peut être désigné pour venir à sa rescousse. Celui-ci va s'occuper des transactions sur une partie des données partagées. Cette allocation dynamique des serveurs permet de s'adapter aux exigences changeantes de certaines applications.

3.2 Organisation de la mémoire

Le système fournit une mémoire virtuelle partagée. Chaque station de travail possède un espace adressable qui lui est propre. Les données manipulées sont des blocs. Un bloc est constitué d'un ensemble d'objets. A propos des données dans le système, nous faisons les hypothèses suivantes:

- chaque bloc mémoire a un nom interne unique reconnu par tous les sites;
- il existe un mécanisme au niveau de chaque site qui identifie de façon unique un bloc mémoire;
- pour tout objet, il existe un mécanisme au niveau de chaque site qui identifie de manière unique le bloc auquel il appartient.

Étant donné que chaque poste de travail a une mémoire locale de capacité limitée, il doit nécessairement exécuter un algorithme de remplacement en cas de faute d'objet. Pour faire un choix judicieux du bloc à remplacer, chaque poste de travail doit avoir un répertoire d'états (RE) comme le montre la figure 3.3 qui identifie l'état des blocs disponibles dans sa mémoire locale.

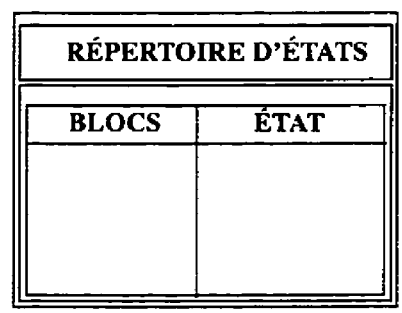
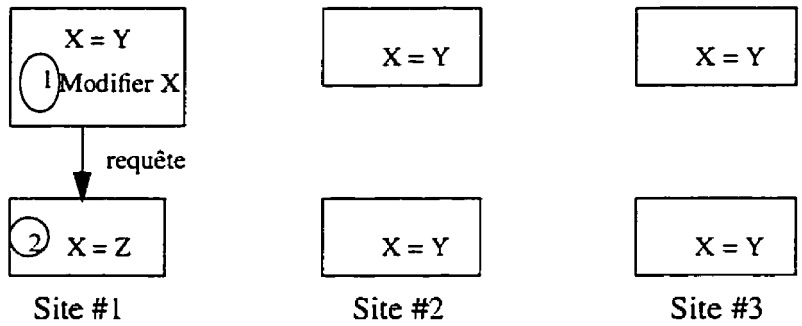


Figure 3.3 Exemple d'un répertoire d'états

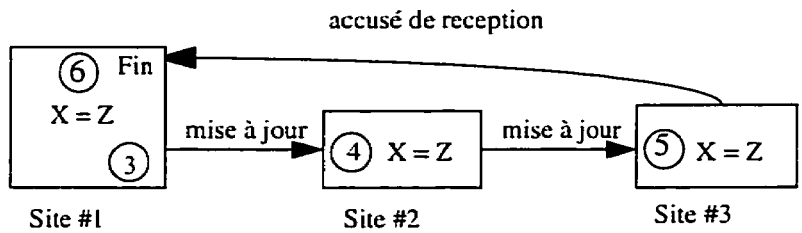
Pour obtenir un objet depuis un poste, il faut d'abord identifier le bloc auquel il appartient dans la mémoire virtuelle de ce poste. Ainsi chaque mémoire locale fonctionne comme une antémémoire vis-à-vis de la mémoire globale.

3.2.1 Réplication des données

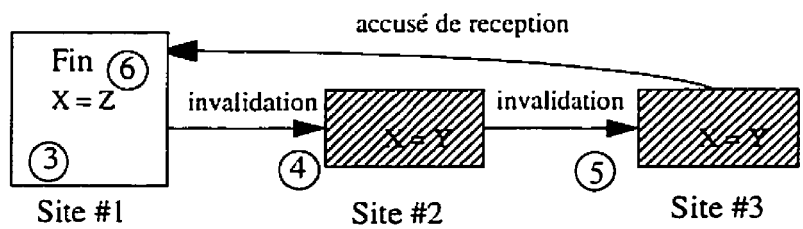
Pour augmenter la disponibilité des données et diminuer leur temps d'accès, il faut les maintenir dans le système en copies multiples. L'existence de copies multiples doit être rendue transparente à l'usager par un algorithme de gestion des copies multiples. Cet algorithme doit être couplé à un mécanisme de contrôle de cohérence pour garantir que toute exécution autorisée est équivalente à une exécution séquentielle des opérations sur une seule copie. La synchronisation induite par l'algorithme de contrôle de cohérence augmente le temps d'attente pour accéder aux données. Cet impact sur le temps d'accès doit être faible pour assurer une certaine efficacité au système. La figure 3.4a représente une donnée incohérente après une opération d'écriture. Ce problème peut être résolu, soit par un mécanisme de mise à jour comme le montre la figure 3.4b, soit par un mécanisme d'invalidation illustré à la figure 3.4c. Dans les deux cas on voit qu'il faut attendre la fin des opérations pour s'assurer que la donnée est cohérente (l'accusé de réception). Cependant, le problème posé est celui de la gestion de ces copies dans le système.



(a) Incohérence de la donnée



(b) Maintient de la cohérence par mise à jour



(c) Maintient de la cohérence par invalidation

Figure 3.4 Problèmes de cohérence avec des copies multiples

3.2.2 Accès et localisation des données

Pour distribuer la charge à travers le système et éviter d'éventuels goulots d'étranglement, il faut permettre une migration des blocs à travers le système. Cette redistribution dynamique des données impose la définition d'un mécanisme pour permettre à tout processus de localiser et d'accéder aux données dont il a besoin. Pour le faire, il faut utiliser un répertoire des copies (RC) disponibles comme le montre la figure 3.5. Il s'agit d'un tableau qui identifie pour chaque bloc les sites où il est disponible.

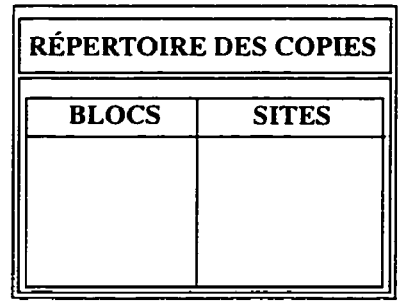


Figure 3.5 Exemple d'un répertoire de copies

Cependant, le problème posé est la localisation de ce dernier. Deux schémas possibles peuvent être utilisés. Le premier préconise l'utilisation de plusieurs répertoires et d'en disposer un par poste. L'avantage de cette méthode est qu'à l'apparition d'une faute d'objet, l'identité du site à consulter est connue localement. D'autre part, en cas de panne d'un poste, il existe toujours une copie du répertoire quelque part dans le système. Le désavantage de cette solution est que tous les répertoires doivent être identiques, c'est-à-dire contenir les mêmes informations. Tout processus qui déplace un bloc doit donc en informer le GCM. Ainsi, après chaque opération de lecture ou d'écriture du RC, il faut tous les mettre à jour. Cette opération est très coûteuse en termes de temps et peut limiter

la bande passante du système surtout s'il est petit, ce qui rend ce schéma inutilisable. Pour réduire le coût des mises à jour, il faut donc réduire le nombre de RC.

L'autre alternative est d'utiliser des répertoires au niveau des serveurs. Ainsi, lors qu'un site veut une information, il n'a qu'à aller consulter le serveur. L'avantage de ce schéma est qu'étant donné le nombre restreint de serveurs comparé au nombre de postes, la pénalité associée aux opérations de mise à jour est moindre. D'autre part la reprise en cas de panne est assurée à cause de la redondance des serveurs. Avec cette solution, il est possible qu'un serveur devienne très sollicité. Cet achalandage peut créer un goulot d'étranglement, d'où l'imposition d'une limite quant à la taille du système. Cependant, le problème peut être résolu en assignant une partie des tâches aux autres serveurs lorsque cette situation survient. Vu la modularité de cette approche et son fonctionnement dynamique, c'est elle qui sera retenue pour notre application.

Les données sont disponibles aux serveurs et dans les mémoires locales des postes. Lors d'une faute d'objet, on peut soit aller chercher la donnée aux serveurs soit à un autre poste. Etant donnée que toutes les transactions passent par les serveurs principaux en cas de faute, il est plus intéressant de faire le chargement à partir de ces derniers. Ceci permet de réduire le temps de réponse à une requête de lecture. Il faut noter que pour chaque bloc, il existe un et un seul serveur principal qui s'occupe de répondre aux requêtes qui l'engagent. Ce choix de design évite une incohérence entre les blocs car deux serveurs ne pourront pas opérer sur une donnée en même temps. Ainsi, les serveurs principaux contiennent des informations différentes. Les serveurs secondaires par contre ont les mêmes informations.

En résumé, un noeud poste de travail est l'illustré à la figure 3.6. On constate qu'il contient un processeur, une antémémoire, une mémoire locale, un répertoire d'états, un répertoire bloc-serveur qui est en fait un RC dont le champ site contient l'identité du serveur principal qui s'occupe de ce bloc, des primitives de synchronisations pour assurer la cohérence des données, des primitives de conversion de format pour des données provenant des postes de nature différentes et enfin d'un tampon pour sauvegarder les requêtes en attente. Enfin, il contient un routeur qui le permet d'utiliser le réseau de communication.

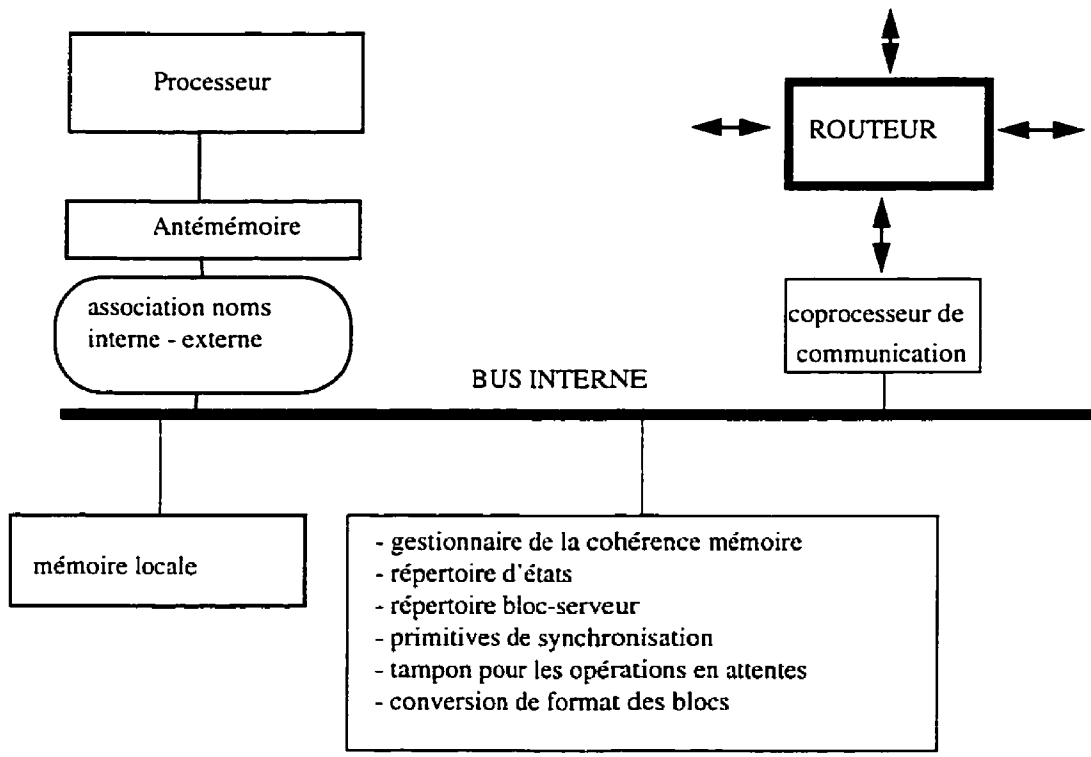


Figure 3.6 Modèle du poste de travail

Un noeud serveur est illustré à la figure 3.7. Il diffère d'un noeud poste de travail par la présence d'un RC. On constate aussi qu'il ne dispose pas d'un RE parce que l'information sur l'état des données est nécessaire seulement au niveau des postes de travail.

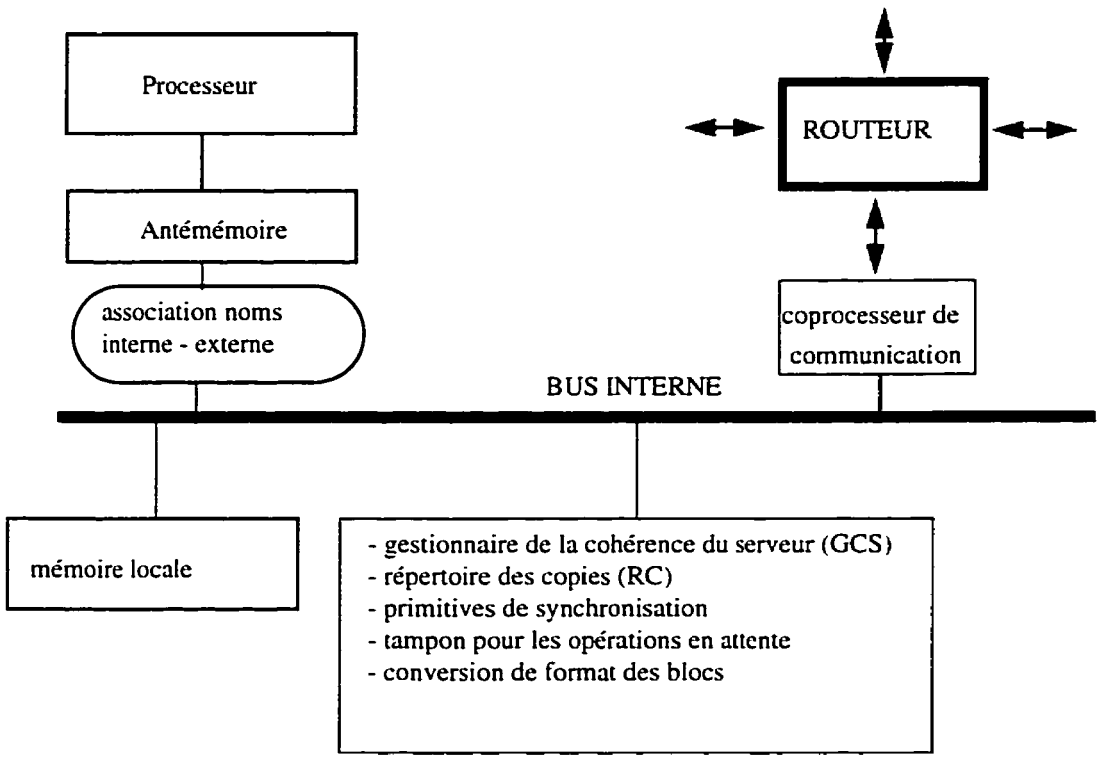


Figure 3.7 Modèle du serveur

3.2.3 Protocoles de cohérence

Comme nous l'avons mentionné précédemment à la section 3.1, le système global peut être décomposé en deux sous-systèmes: le sous-système des postes de travail qui sert à exécuter les programmes parallèles, et le sous-système des serveurs qui sert à la sauvegarde des données. D'après le chapitre précédant, nous disposons de deux familles de protocoles: la première est la famille des protocoles d'invalidations en écriture et la seconde la famille des protocoles de mise à jour en écriture. Le modèle des postes de travail peut supporter les deux familles de protocoles. Le modèle des serveurs pour sa part

nécessite uniquement le protocole de mise à jour car les informations doivent être identiques dans tous les SS. Ainsi au niveau du modèle des poste de travail, nous pouvons faire deux combinaisons de protocoles qui nous donne deux modèles physiques identiques mais deux modèles logiques différents. Le premier modèle logique est le modèle d'invalidation en écriture au niveau des postes et de mise à jour au niveau des serveurs et le second est le modèle de mise à jour en écriture au niveau des postes et de mise à jour des serveurs. À cause de ces différences au niveau du fonctionnement logique, la modélisation mathématique de ces deux modèles sera différente. Dans la suite de ce chapitre, nous allons décrire les différents algorithmes de gestion de copies multiples que nous utilisons dans nos modèles.

3.3 Protocole de mise à jour en écriture

Il s'agit d'énoncer le mécanisme permettant de maintenir une cohérence entre les données. Ce protocole adopte l'hypothèse selon laquelle toutes les copies partagées sont accessibles par tous les postes. Dans ce protocole, une opération de lecture se traduit par l'accès à une seule copie alors que lors d'une requête d'écriture, il faut obligatoirement mettre à jour toutes les autres copies disponibles du système.

Une variante de cet algorithme est celui du propriétaire de la donnée. Les écritures sont toujours autorisées par le propriétaire et porte d'abord sur sa copie. Dans ce schéma, seul le propriétaire de la donnée a le droit de modifier la donnée et d'autoriser une mise à jour des autres copies. Ainsi, pour modifier une donnée, il faut au préalable obtenir le privilège. Pour devenir propriétaire, il faut soit être le premier à accéder à la donnée, soit vouloir la modifier. Avec ce schéma, chaque donnée a en tout temps un seul propriétaire. Un autre

avantage de cette approche est que le choix du propriétaire est transparent et le propriétaire varie de manière dynamique et en fonction du programme qui s'exécute. Ceci permet d'adapter le système à l'application qu'il traite. Cette approche permet aussi de synchroniser de manière implicite les opérations car si par exemple un processus veut accéder à une variable au moment où elle est modifiée, alors ce conflit peut être directement détecté au niveau du serveur en consultant seulement l'état du propriétaire de la donnée au lieu de toutes les copies de la donnée et de ralentir le déroulement de cette transaction. Elle permet également d'avoir un site pour recevoir les accusés de réception afin de déterminer la fin d'une opération d'écriture. Notre algorithme sera calqué sur ce second schéma.

L'état d'un bloc dans une mémoire locale peut être dans l'un des trois choix suivants:
ABSENT (ABS): c'est-à-dire non présent dans cette mémoire. Pour obtenir une copie du bloc, il faut aller au SP.
LECTURE SEULEMENT (RO): c'est-à-dire que ce site n'est pas le propriétaire du bloc. Pour que ce poste puisse le modifier, il faut au préalable obtenir le privilège d'écriture du propriétaire. Il faut noter que cet état nous indique également qu'il y a au moins deux copies de ce bloc dans le système.
LECTURE ÉCRITURE (RW): c'est-à-dire que ce site est propriétaire du bloc et possède donc les privilèges sur celui-ci. Il peut effectuer des opérations de lecture et d'écriture sur le bloc sans consulter les autres. Cet état nous indique également qu'on peut avoir un ou plusieurs exemplaires du même bloc dans le système.

La figure 3.8 montre les états et les opérations qu'il faut effectuer pour faire une transition. Il faut noter que c'est le serveur principal qui fournit le bloc.

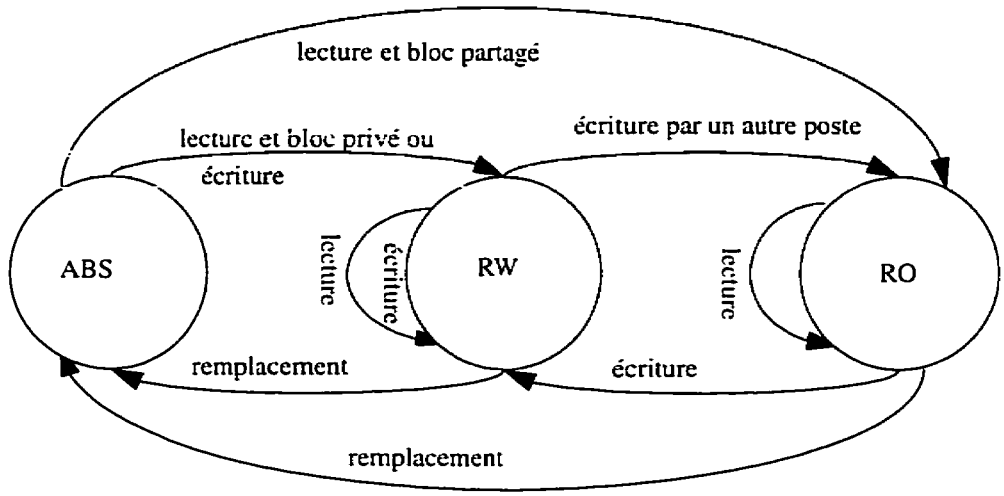


Figure 3.8 Diagramme de transition d'état d'un bloc en mémoire locale

3.3.1 Scénarios des transactions à travers le système

Accès d'un bloc en lecture

Si le processeur réclame un bloc et que celui-ci est dans la mémoire locale à l'état RO ou RW, alors le GCM transfère la donnée au processeur. L'état du bloc reste le même et il n'y a pas d'accès au réseau pour réaliser l'opération.

Si par contre le bloc est dans l'état ABSENT, alors il s'agit d'une faute de bloc. Le GCM identifie le bloc, formule une requête et l'envoie au SP en utilisant le réseau de communication et crée de l'espace pour recevoir le bloc en exécutant un algorithme de remplacement. Dès qu'il reçoit la réponse à sa requête, il met à jour l'état du bloc dans son RE, écrit le bloc en mémoire et transfère le bloc au processeur. Lorsque le GCS reçoit

cette requête de lecture, il transfère le bloc ainsi que son état au demandeur, met à jour son RC et met à jour le RC des SS. Si au niveau du SP il s'agit de la seule copie distribuée dans le réseau, alors son nouvel état est RW sinon il est RO. La figure 3.9 illustre le cas où l'état du bloc du demandeur transite de ABS à RW, et la figure 3.10 montre le cas où son état passe de ABS à RO. Dans le premier cas, le demandeur devient propriétaire de la donnée et peut la modifier sans contrainte. Dans le second cas, le demandeur ne peut modifier le bloc qu'après avoir obtenu au préalable le privilège d'écriture.

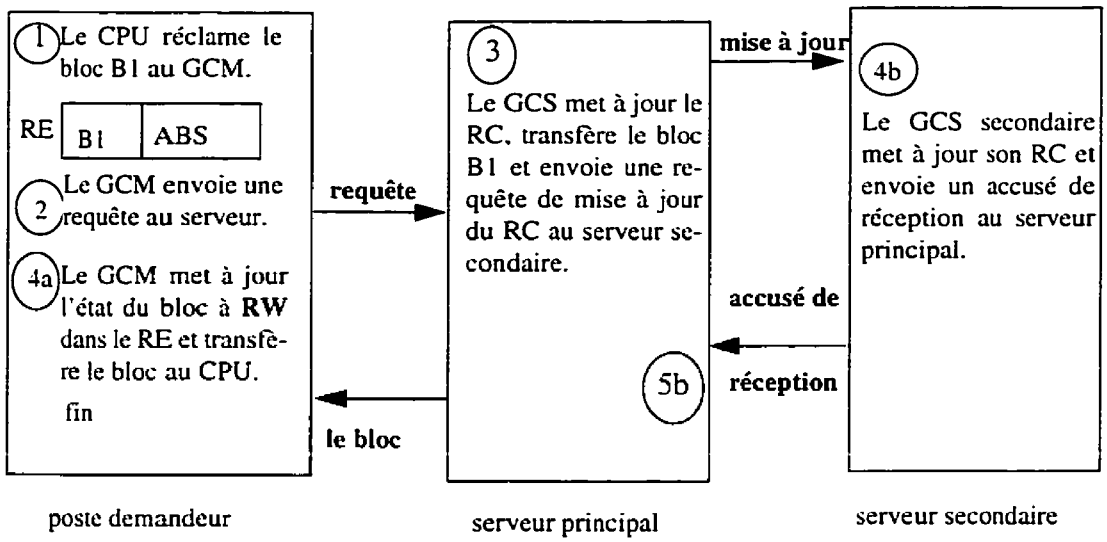


Figure 3.9 Obtention d'un bloc

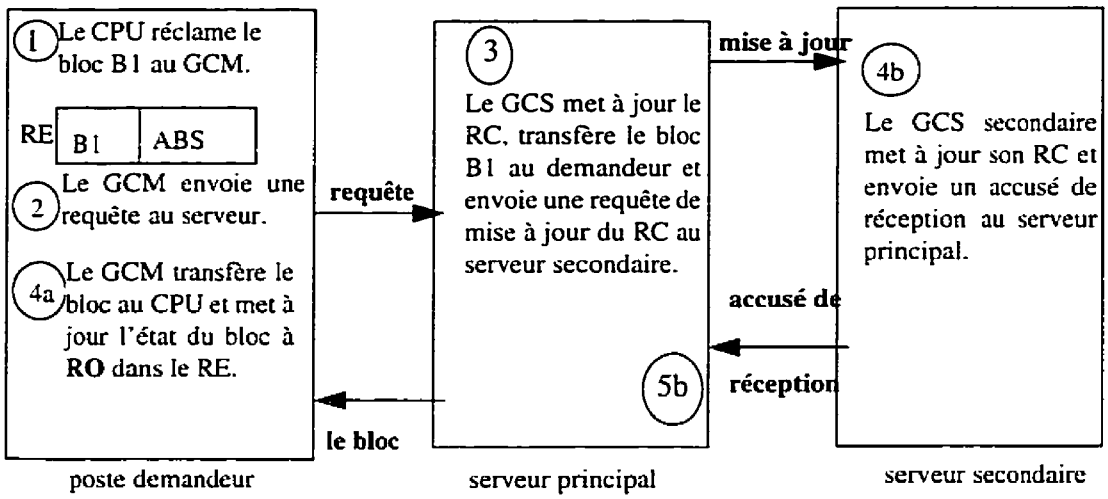


Figure 3.10 Obtention d'un bloc avec copie existante

Accès en écriture d'un bloc en mémoire

Si le processeur veut modifier un bloc et que celui-ci est dans sa mémoire locale, alors deux scénarios sont possibles.

- Si le bloc est dans l'état RW, le GCM autorise le processeur à modifier le bloc. Une fois le bloc modifié, il transmet une requête au SP contenant une copie modifiée du bloc ainsi qu'une demande de mise à jour de toutes les autres copies du système y compris celle qui demeure au SP. Le serveur pour sa part met à jour sa copie. Selon le nombre de copies dans le réseau, l'opération peut suivre un des deux scénarios. Si aucun autre poste n'a une copie de la donnée, alors le SP transmet immédiatement l'accusé de réception au demandeur et par la suite met à jour les copies des SS. Par contre s'il existe des copies dans d'autres postes, alors le SP va mettre à jour toutes ces copies. Après avoir terminé cette opération, il envoie un accusé de réception au demandeur. Par la suite, il met à jour les copies disponibles aux SS, et l'état du bloc reste le même.

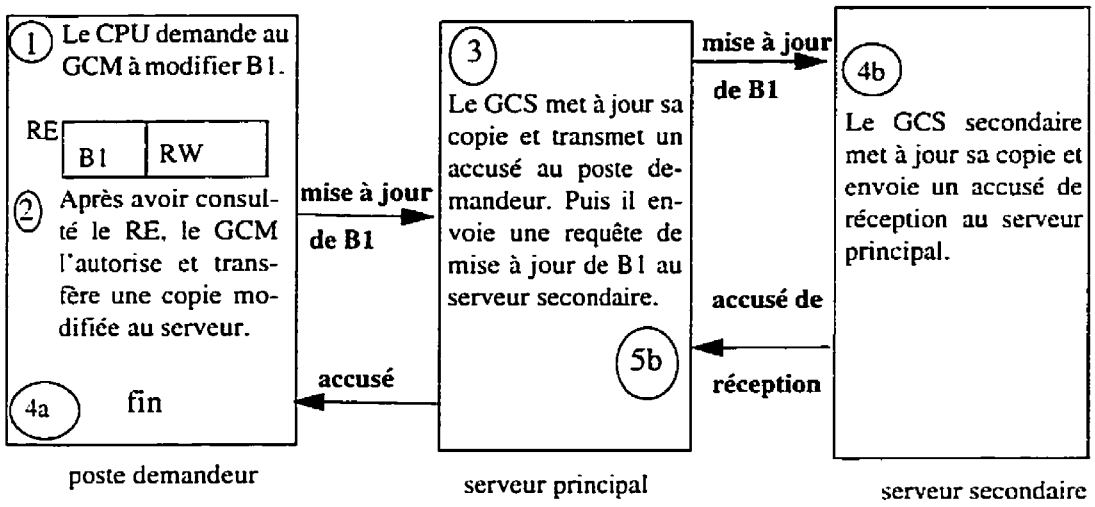


Figure 3.11 Modification d'un bloc RW

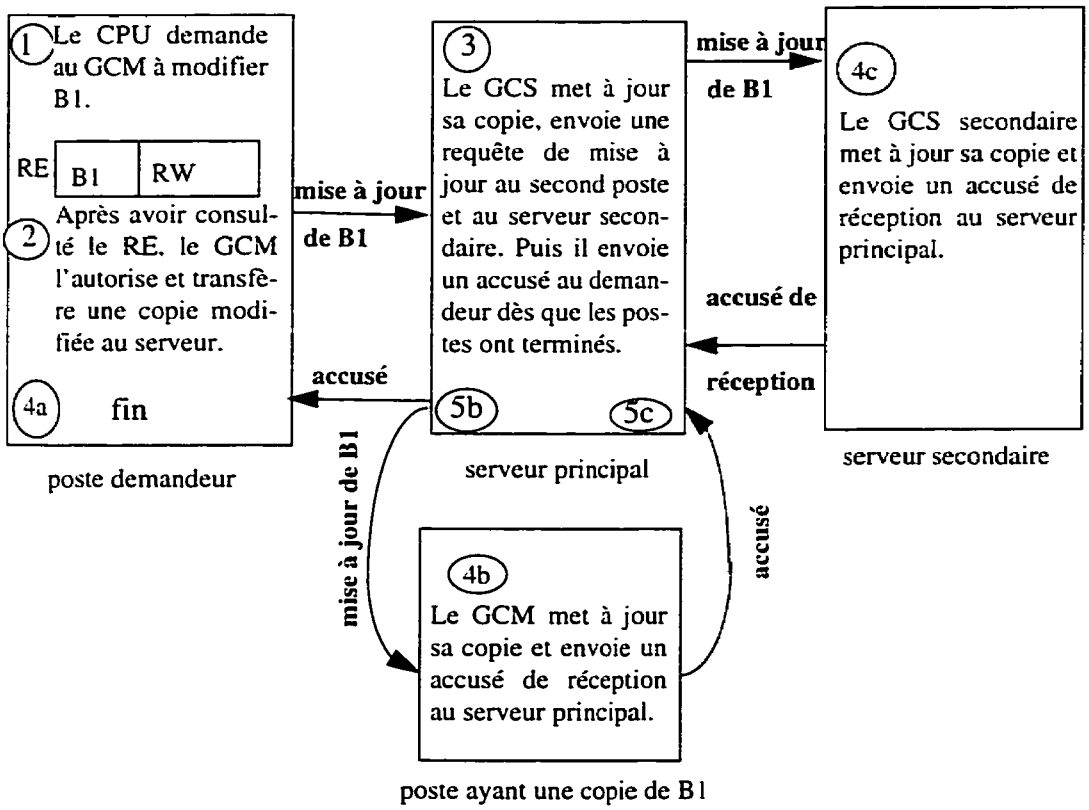


Figure 3.12 Modification d'un bloc RW avec copie RO existante

La figure 3.11 illustre le cas où le bloc n'est pas partagé par un autre poste et la figure 3.12 le cas où il est partagé. Dans les deux cas, la transaction sera terminée dès que le demandeur reçoit l'accusé du SP. Si le bloc est dans l'état RO, alors le GCM envoie une requête au SP afin d'obtenir le privilège d'écriture. Après avoir reçu la requête, le SP transfère le privilège au demandeur et en même temps transmet un message à l'expropriétaire pour l'informer de sa perte de privilège. Dès que le demandeur obtient le privilège, il met à jour son RE et son GCM autorise le processeur à modifier le bloc. Une fois le bloc modifié, le demandeur transmet une requête au SP contenant une copie modifiée du bloc ainsi qu'une demande de mise à jour de toutes les autres copies du système y compris celle qui demeure au SP. Le SP pour sa part met à jour sa copie et ensuite il met à jour toutes les copies aux postes. Après avoir terminé cette opération, il envoie un accusé de réception au demandeur. Par la suite, il met à jour les copies disponibles aux SS. Au niveau du demandeur, l'état du bloc passe de RO à RW, alors qu'au niveau de l'expropriétaire l'état devient RO.

Les figures 3.13a et 3.13b illustrent en détail les étapes suivies et les différentes transactions transmises entre les différents sites impliqués. Dans cette opération, il faut d'abord obtenir du SP le privilège selon la figure 3.13a et modifier le bloc selon la figure 3.13.b

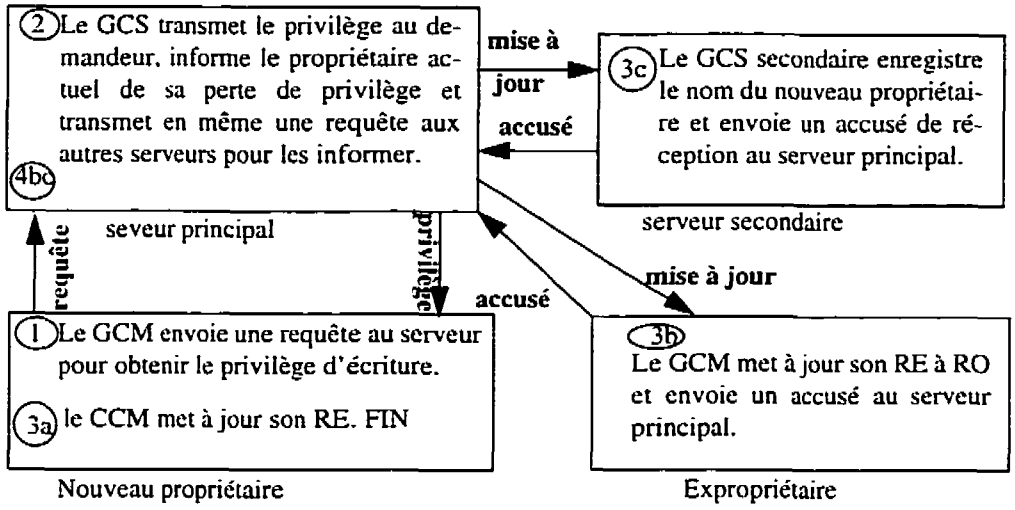


Figure 3.13a Obtention d'un bloc et du PE

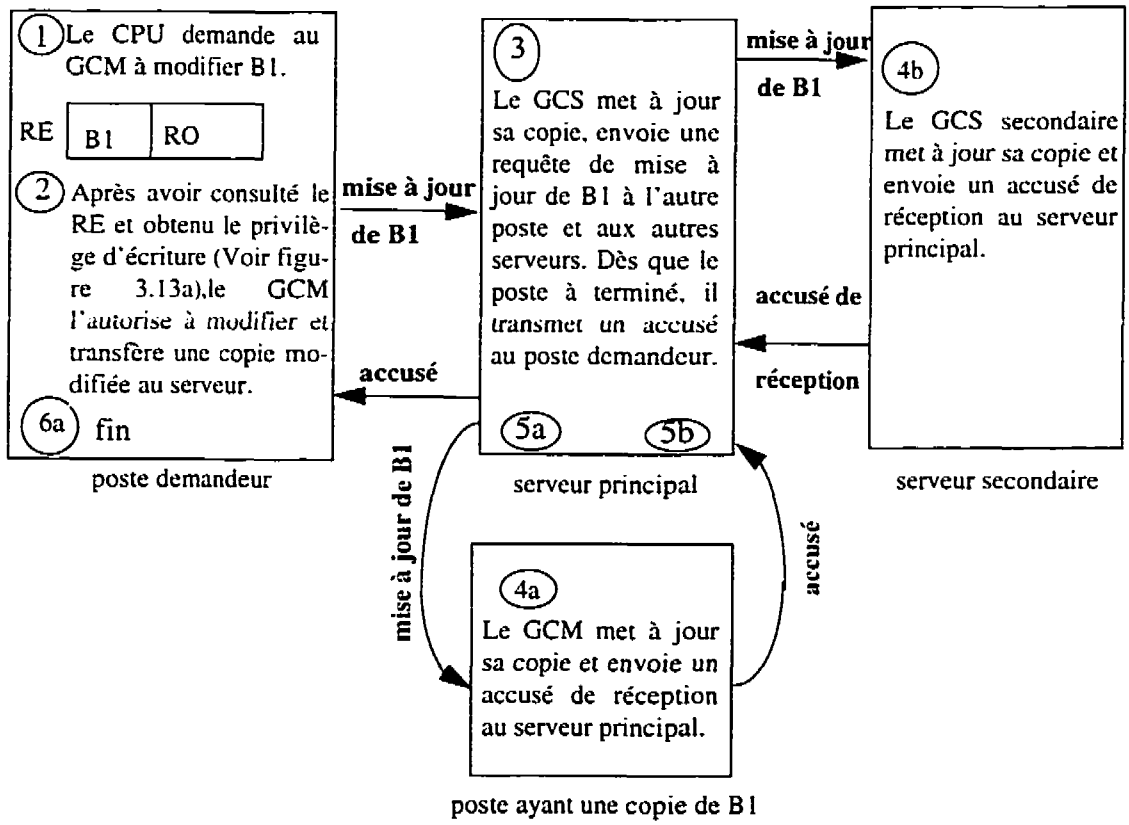


Figure 3.13b Modification d'un bloc RO avec copie existante

Accès en écriture d'un bloc absent en mémoire

Si par contre le bloc est à l'état ABS, alors une requête est formulée et transmise au SP lui demandant une copie du bloc ainsi que le privilège d'écriture. Au niveau du serveur, deux scénarios sont encore possibles.

- S'il s'agit de la première fois qu'un poste réclame le bloc, alors le SP enregistre l'identité du nouveau propriétaire dans son RC et transfère la donnée ainsi que le privilège au demandeur comme le montre la figure 3.14a. Par la suite il met à jour le RC des SS. Lorsque le demandeur reçoit la réponse à sa requête, il met à jour son RE, écrit le bloc en mémoire et autorise le processeur à le modifier. Une fois cette opération terminée, il renvoie une requête au SP contenant la copie modifiée afin que le serveur puisse mettre sa copie à jour. Le SP qui reçoit cette requête met à jour sa copie et envoie un accusé de réception au demandeur. Par la suite il met à jour les copies des SS comme le montre la figure 3.14b. Au niveau du poste demandeur, l'état du bloc passe de ABS à RW.

- S'il ne s'agit pas de la seule copie du bloc dans le réseau, alors le SP procède comme à la figure 3.15a. Il enregistre l'identité du nouveau propriétaire transfère le bloc ainsi que le privilège au demandeur et informe l'expropriétaire de sa perte de privilège. Par la suite il met à jour le RC des SS. Lorsque le demandeur obtient le bloc et le privilège, il procède comme le montre la figure 3.15b. Il met à jour son RE, écrit le bloc en mémoire et autorise le processeur à le modifier. Dès que la modification est terminée, il envoie une requête au SP contenant la copie modifiée. Lorsque le SP reçoit cette requête, il met à jour sa copie ainsi que toutes les autres copies valides aux postes. Puis, il envoie un accusé de réception au demandeur. Par la suite il met à jour les copies des autres postes. Au niveau du

demandeur, l'état de la donnée passe de ABS à RW alors qu'au niveau de l'expropriétaire, l'état devient RO.

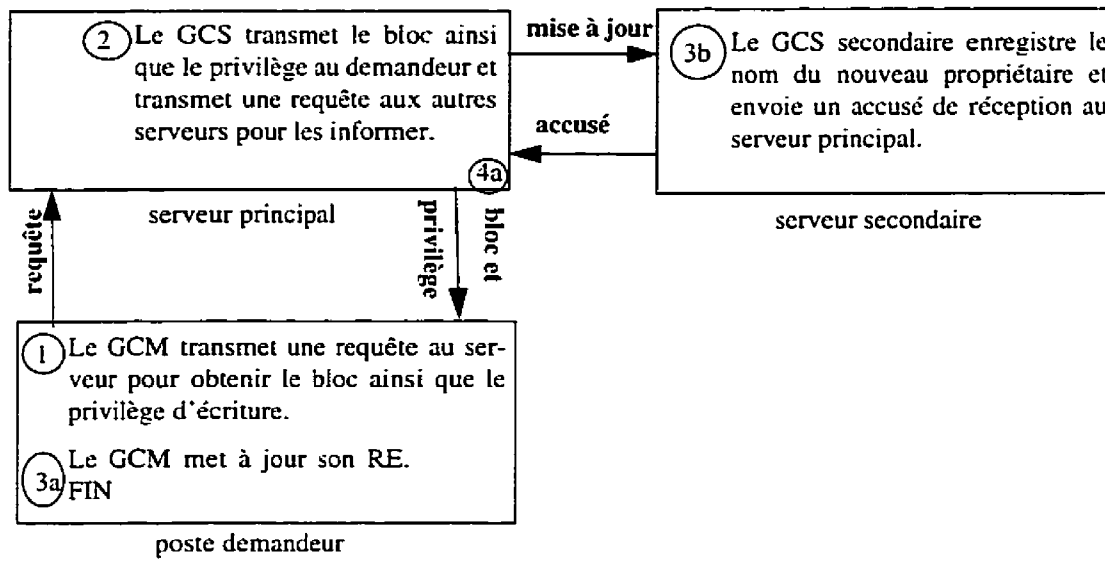


Figure 3.14a Obtention du bloc unique et du PE sans copies aux postes

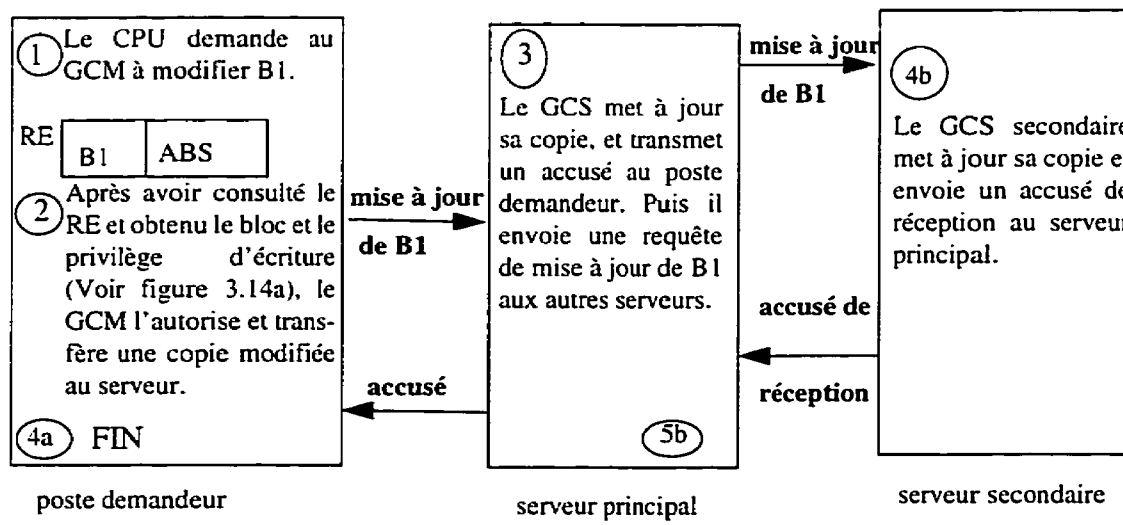


Figure 3.14b Modification d'un bloc ABS sans copie existante

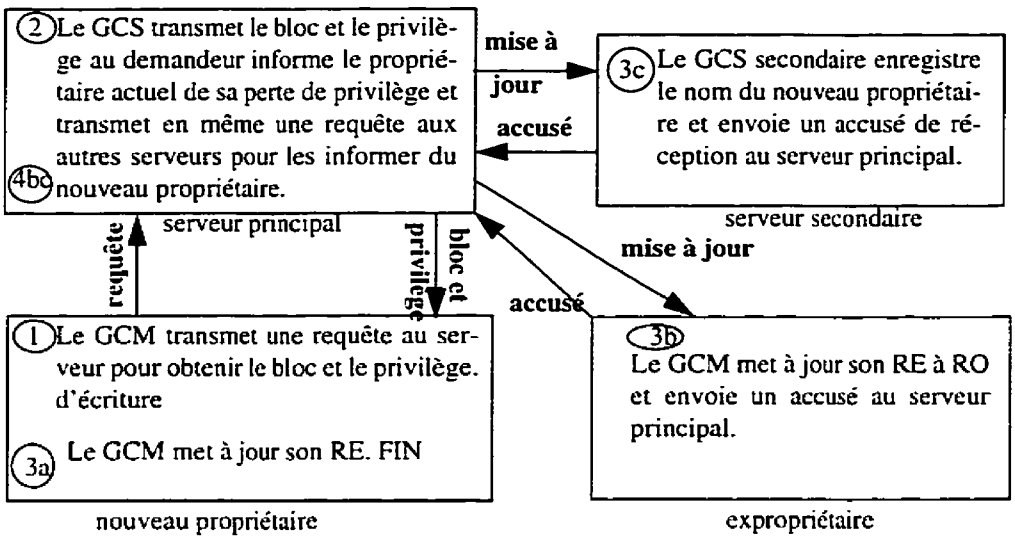


Figure 3.15a Obtention d'un bloc ABS et du PE avec copie RW disponible

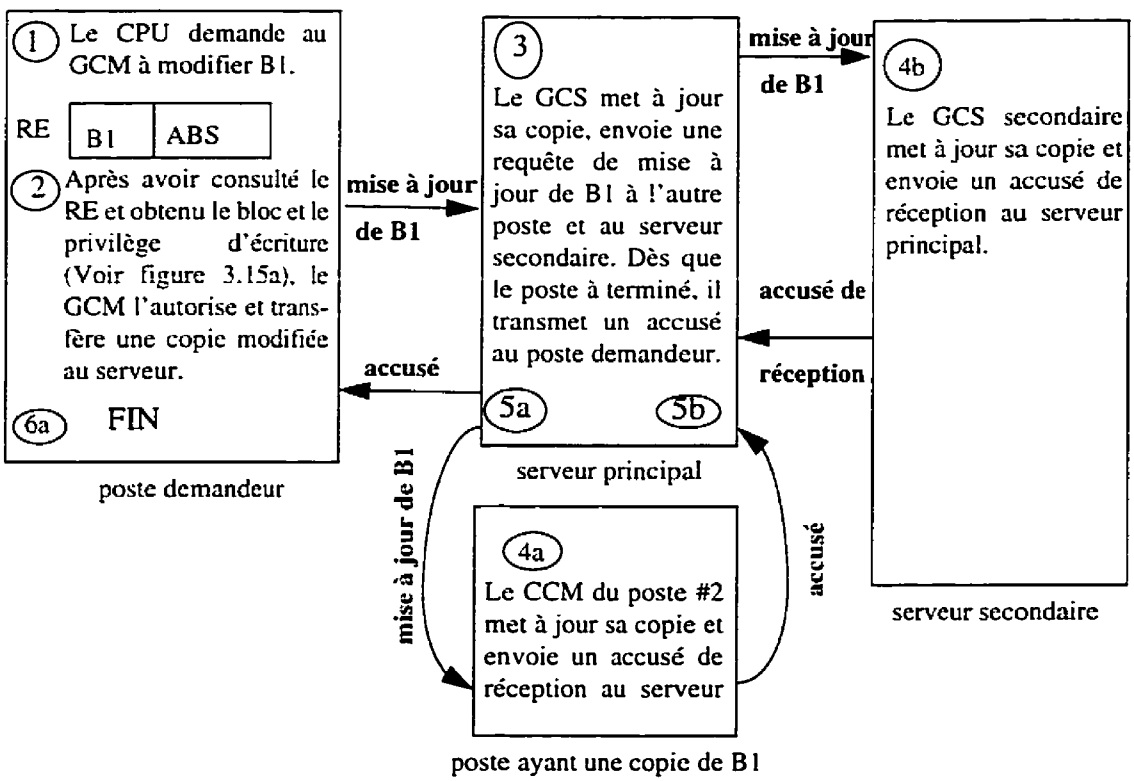


Figure 3.15b Modification d'un bloc ABS avec copie existante

Traitement en cas de remplacement d'un bloc

Chaque fois qu'un bloc ayant le privilège d'écriture est choisi par un algorithme de remplacement, alors une requête doit être envoyée au SP lui demandant de transférer le privilège à un des postes possédant une copie du bloc. Le choix se fait de manière aléatoire. Si le bloc n'est pas disponible ailleurs, alors le SP devient le nouveau propriétaire.

3.4 Protocole de mise à jour des serveurs

Dès que le SP enregistre une information provenant des postes de travail, il met à jour les SS. La nature de ces informations peut être une donnée qui vient d'être modifiée, une donnée qui vient de changer de site, ou encore un transfert de privilège. Tous ces attributs constituent l'état de la donnée et sont nécessaires pour une reprise en cas de panne. Le serveur peut procéder à cette opération de deux façons: soit par diffusion soit par chaînage.

La diffusion est généralement utilisé lorsqu'un processeur veut faire connaître à plusieurs autres une information qu'il détient localement. Dans le cas précis, le mécanisme de fonctionnement est le suivant: le serveur envoie les informations en une seule requête aux autres et attend d'eux des accusés de réception individuels. L'opération se termine dès que le nombre d'accusés de réception escompté est obtenu. Cette technique est présentée à la figure 3.16 où une information de mise à jour doit être diffusée à deux serveurs. Le principe est le même quelque soit la nature de la requête et le nombre de serveurs impliqués. Il faut noter que dans le cas où c'est le réseau réalise la diffusion, nous

parlerons de diffusion véritable ou tout simplement de diffusion. Dans le cas où c'est le programmeur qui la réalise, nous parlerons de diffusion simulée.

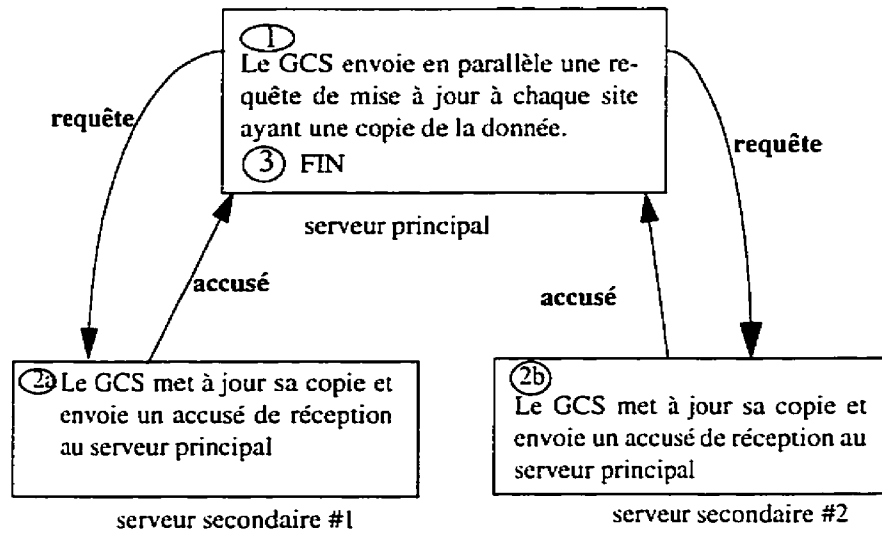


Figure 3.16 Mise à jour par diffusion

Le chaînage est utilisé pour les mêmes fins que la diffusion alors que son fonctionnement diffère. Dans le cas qui nous concerne, l'opération de mise à jour est la suivante: le SP envoie les informations à un SS ainsi que la liste de tous les SS par où les informations doivent transiter. Le nom du SP est le dernier sur cette liste. Dès que le premier SS reçoit cette requête, il met à jour ses données, enlève son nom de la liste des serveurs à consulter et transfère la requête au prochain sur la liste. Le prochain procède de manière identique et étant donnée que le dernier de la liste est le serveur principal, la requête lui revient ce qui est pour lui un accusé que la transaction s'est bien déroulée. Dans cette opération, l'accusé de réception est collectif. Cette technique est présentée à la figure 3.17 avec deux SS.

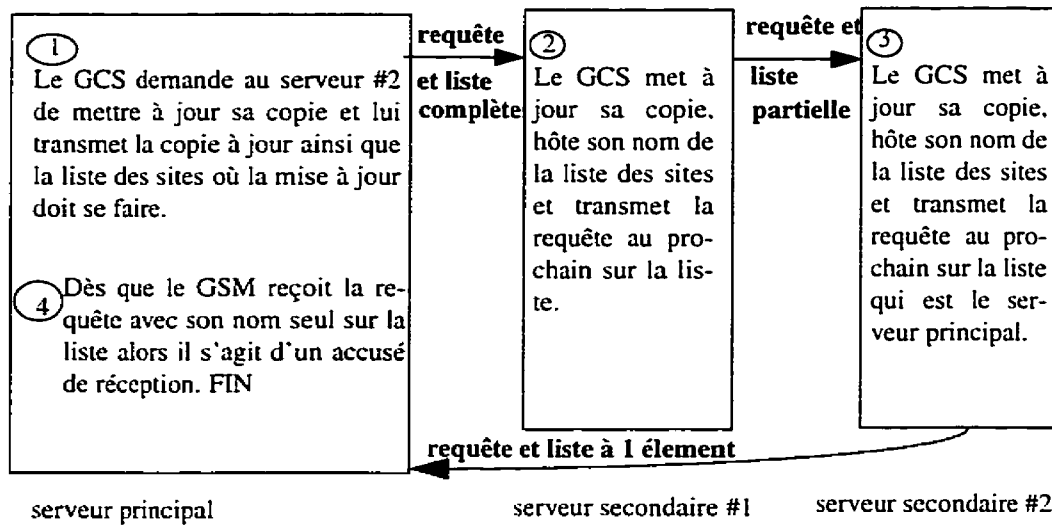


Figure 3.7 Mise à jour par chaînage

Ces deux techniques de mise à jour peuvent donner des résultats différents. Nous allons donc analyser leurs effets sur les performances du système. Ainsi pour ce modèle nous avons deux sous-modèle qui sont identiques au niveau des postes mais diffèrent au niveau des serveurs.

3.5 Protocole d'invalidation en écriture

Le protocole que nous décrivons ci-après adopte l'hypothèse que toute copie partagée est accessible par tous les autres postes. Ce protocole réalise une opération de lecture en consultant une seule copie alors que pour une écriture, il invalide préalablement toutes les copies disponibles sauf celle qui sera modifiée. Le protocole peut être couplé à une cohérence libre. Cependant, le protocole que nous allons utiliser obéit aux mêmes

principes que l'algorithme du propriétaire que nous avons décrit dans la section 3.3.

L'état d'un bloc dans une mémoire locale peut être dans l'une des quatre catégories suivantes:

L'état ABSENT (ABS) indique que la donnée n'est pas présente dans cette mémoire. Pour obtenir une copie du bloc, il faut aller au serveur.

L'état LECTURE SEULEMENT (RO) signifie que le site n'est pas le propriétaire de la donnée. Pour que le poste puisse la modifier, il faut qu'il obtienne au préalable le privilège d'écriture du propriétaire actuel. Il faut noter que cet état nous indique indirectement qu'il y a au moins deux copies du bloc dans le système.

L'état LECTURE ÉCRITURE (RW) signifie que le site est propriétaire du bloc et qu'il possède tous les privilèges sur celui-ci. Il peut effectuer des opérations de lecture et d'écriture dans le bloc sans consulter les autres. Cet état ne nous indique pas le nombre d'exemplaires dans le système, omis du fait qu'il réside dans le poste. Pour avoir une copie dans cet état, il faut être le premier à la charger ou à le modifier. Après une opération d'écriture, seul le poste de travail initiateur de la requête possède le bloc.

L'état INVALIDE (INV) indique que le poste a reçu un message d'invalidation et qu'une copie dans l'état RW existe dans le système.

3.5.1 Scénario des transactions à travers le système

Accès d'un bloc en lecture

Si le processeur réclame un bloc et que celui-ci est dans sa mémoire locale dans l'état RO ou RW, alors le GCM transfère le bloc au processeur. L'état du bloc reste le même.

Si par contre l'état du bloc est ABS ou INV, alors l'accès provoque une faute de bloc. Le GCM identifie le bloc, formule une requête, le transmet au SP en utilisant le réseau de communication et crée de l'espace mémoire pour recevoir le bloc. Dès qu'il obtient la réponse à sa requête, il transfère une copie du bloc au processeur, enregistre l'état du bloc dans son RE et écrit le bloc en mémoire.

Lorsque le SP reçoit la requête de lecture, le GCS transfère immédiatement le bloc ainsi que son état au demandeur, il met à jour son RC et transmet cette information aux SS. Si le bloc est l'unique exemplaire dans les postes, c'est à dire qu'il s'agit du premier accès au bloc, l'état enregistré est RW sinon il est RO.

Accès d'un bloc en écriture

Si le processeur veut modifier un bloc et que celui-ci est dans sa mémoire locale, alors deux scénarios sont possibles.

Si le bloc est dans l'état RW alors le GCM identifie le serveur responsable du bloc et lui transmet une requête pour invalider toutes les copies valides dans les postes. Dès qu'il obtient la quittance à sa requête, il autorise le processeur à modifier le bloc. Lorsque cette opération est terminée, il transmet une copie modifiée au serveur pour lui permettre de mettre à jour sa copie et attend de lui un accusé de réception.

Lorsque le SP reçoit une requête d'invalidation, il procède de deux façon selon le nombre de copies du bloc dans le réseau. Si le bloc est disponible aux serveurs uniquement, alors

le GCM du SP transfère immédiatement un accusé au demandeur. Si le bloc est disponible aux postes, alors le GCS envoie un message d'invalidation à tous les postes ayant une copie valide. Dès que l'opération est terminée, il transmet un accusé au demandeur. Par la suite il envoie un message de mise à jour de son RC aux SS.

Lorsque le SP reçoit une requête de mise à jour, il met à jour sa copie et transmet un accusé de réception au demandeur. Par la suite, il envoie un message de mise à jour aux SS. Au niveau du demandeur, l'état du bloc reste le même mais au niveau des autres postes, il passe de RO à INV.

Par contre, si le bloc dans le site effectuant un accès en écriture est dans l'état RO, le GCM envoie une requête au SP pour invalider toutes les autres copies valides et obtenir le privilège d'écriture. Dès qu'il obtient le privilège, il met à jour son RE et autorise le processeur à modifier le bloc. Lorsque cette opération est terminée, il transmet une copie modifiée au SP pour lui permettre de mettre à jour sa copie et attend de lui un accusé de réception. Lorsque le serveur reçoit une requête de transfert de privilège et d'invalidation, il envoie un message d'invalidation à tous les postes ayant une copie du bloc. Dès que cette opération est terminée, il transfère le privilège au demandeur. Par la suite, il envoie un message contenant la mise à jour de son RC à tous les SS. Dès qu'il reçoit une requête de mise à jour, il procède comme dans le cas précédant. Au niveau du poste demandeur l'état du bloc passe de RO à RW et au niveau des autres postes, l'état devient INV.

Si par contre le bloc dans le site demandeur est dans l'état ABS ou INV, alors une requête est formulée et transmise au serveur lui demandant une copie du bloc ainsi que le privilège d'écriture. Dès qu'il reçoit la réponse à sa requête, il met à jour son RE, écrit le bloc en

mémoire et autorise le processeur à modifier le bloc. Lorsque cette opération est terminée, il transmet une copie modifiée au SP pour lui permettre de mettre à jour sa copie et attend de lui un accusé de réception. Lorsque le SP reçoit la requête pour le transfert de la donnée et du privilège, il procède de deux façons selon le nombre de copies existant dans le réseau. Si aucun poste ne possède le bloc, le serveur transfère la donnée ainsi que le privilège au demandeur. Si le bloc est disponible ailleurs, alors il invalide toutes les copies disponibles dans les postes. Dès que l'opération est terminée, il envoie le bloc ainsi que le privilège au demandeur. Par la suite, il met à jour le RC de tous les SS. Lorsque la requête est une mise à jour, il procède comme dans le cas précédant. Au niveau du site demandeur, le nouvel état de la donnée devient RW alors qu'au niveau des autres postes, leurs états passe à INV.

Traitement en cas de remplacement d'un bloc

Chaque fois qu'un bloc avec le privilège d'écriture est choisi par l'algorithme de remplacement, une requête doit être envoyée au SP lui demandant de transférer le privilège du propriétaire à un des postes possédant une copie du bloc. Le choix se fait de manière aléatoire. Si le bloc n'est pas disponible ailleurs, alors le SP redevient le nouveau propriétaire tel qu'il l'a été lorsque le programme réparti a débuté.

CHAPITRE 4

MODÈLES MATHÉMATIQUES

Dans le chapitre précédent, nous avons présenté les différents modèles à analyser. Dans le présent chapitre, nous allons tenter d'obtenir de ceux-ci les modèles analytiques et proposer une façon de les résoudre. Pour le faire, il faut d'abord poser les hypothèses sur les modèles, caractériser et quantifier les paramètres de ces derniers. Étant donné que nous avons deux modèles différents, nous allons les étudier à tour de rôle. Nous utiliserons les mots objet et bloc pour désigner l'unité sémantique des données en mémoire. Autrement dit objet et bloc sont synonymes.

4.1 Hypothèses des modèles

Pour l'analyse des modèles, nous faisons les hypothèses suivantes:

Hypothèse 1: Les postes de travail sont identiques; c'est à dire que l'environnement est homogène. Cette hypothèse est généralement vraie pour les systèmes répartis faiblement couplés. Par contre pour les grands systèmes, il est possible de trouver des composants d'origines différentes, car ils sont très souvent construits par l'interconnexion de systèmes faiblement couplés. Dans ce cas, nous supposons que cette hypothèse n'est plus valable.

Hypothèse 2: Les postes de travail sont statistiquement indépendants. C'est une propriété implicite lors de la spécification d'un système réparti. Ses composants sont toujours autonomes.

Hypothèse 3: Le système est asynchrone en ce sens qu'il n'y a pas d'horloge globale qui diffuse son heure à tous les autres postes. C'est généralement le cas des systèmes répartis. Les postes ont un fonctionnement interne synchrone mais la communication entre les postes se fait à travers le réseau. Compte tenu de l'utilisation de ce dernier il est possible que le temps de réponse soit variable.

Hypothèse 4: Le temps d'accès au médium de communication est fini. Cette propriété est implicite lors de la spécification d'un système réparti, car tous les postes peuvent communiquer entre eux.

Hypothèse 5: Les mémoires locales ont une capacité finie. Les postes de travail sont toujours caractésisés par la capacité de leurs mémoires. Leurs tailles sont de plus en plus grandes, mais hélas ne peuvent pas contenir toutes les données nécessaires par tous les programmes. Ainsi, tous les systèmes répartis sont toujours munis d'un algorithme d'allocation de mémoire pour diminuer le nombre de fautes de pages [Krakowiak 87].

Hypothèse 6: Les données sont indépendantes et ont les mêmes tailles. En effet, plusieurs systèmes organisent leurs mémoires partagées en blocs de tailles fixes et identiques. Par contre, l'hypothèse d'indépendance n'est pas très réaliste car les fautes d'objet sur une donnée dans une mémoire influencent les autres données qui y sont. Par exemple lors du chargement d'un bloc à partir du serveur, il faut remplacer un autre bloc si la mémoire est pleine. Cependant, nous supposons que ces influences mutuelles entre les données sont identiques et uniformément distribuées sur l'ensemble des données en mémoire.

Hypothèse 7: Les références sont uniformes. Cette hypothèse stipule que tous les blocs du système ont la même probabilité d'être adressés par une requête. Cette hypothèse va à l'encontre de la propriété de la non-uniformité des chaînes de références dans un programme [Krakowiak 87]. Cependant, si on choisit un ensemble de travail dont la fenêtre d'observation est très petite, alors cette hypothèse reste valable.

Hypothèse 8: Les références aux blocs sont uniques. Ainsi, une requête ne peut être destinée qu'à un et un seul bloc, ce qui est généralement le cas dans les programmes si on prend des instructions atomiques.

Hypothèses 9: Il y a un serveur principal par bloc. Cette hypothèse stipule que tout bloc du système à un serveur principal unique qui gère les requêtes qui lui sont destinées; ce qui est en pratique très plausible, si on considère des problèmes de synchronisation et les coûts associés à l'utilisation de plusieurs serveurs par bloc.

Hypothèse 10: Les serveurs ont une capacité infinie. Ce qui veut dire qu'ils sont assez grands pour contenir toutes les données partagées. Cette hypothèse est généralement vraie lorsqu'on utilise des serveurs de fichiers ou de base de données. Cependant s'il faut en utiliser plusieurs par manque de place, alors on les regroupe et on les considère comme une seule entité.

Hypothèse 11: Le système est fiable. En cas de panne, nous considérons que la reprise est immédiate, c'est à dire sans délai. Généralement les systèmes répartis fonctionnent très bien. Cependant en cas de panne, il faut attendre un délai supplémentaire dû à l'exécution du logiciel de reprise.

Hypothèse 12: Le service de diffusion peut être offert soit par le système d'exploitation du réseau, soit par l'usager. Dans le premier cas nous parlerons de diffusion véritable ou tout simplement de diffusion. Dans le second cas, nous parlerons de diffusion simulée.

Hypothèse 13: Nous supposons que le temps d'interarrivée des requêtes suit une distribution exponentielle. Nous supposons ausi que les temps de service aux mémoires locales, au réseau et aux serveurs suivent également une distribution exponentielle sauf indication contraine.

4.2 Modèle à mise à jour

Le fonctionnement logique du système implique deux entités dont les processeurs et les données. Les processeurs effectuent des références sur les données. Ils sont donc actifs. Les données par contre subissent les actions des processeurs. Elles sont donc passives. Vu que ces deux entités ont des comportements diamétralement opposés, nous allons les analyser séparément.

4.2.1 Modèle des processeurs

Il s'agit dans cette section de modéliser le comportement des processeurs à partir du système physique et de son comportement logique. En utilisant les hypothèses 1 et 2, nous allons restreindre cette analyse à un seul processeur. Le modèle obtenu est identique pour tous les autres processeurs.

4.2.1.1 Description du modèle

1- Caractérisation des paramètres du modèle

Pour caractériser le modèle, nous allons nous baser sur le comportement du processeurs. Ses activités peuvent être de trois types:

- calculs;
- références aux variables partagées disponibles en mémoire locale;
- références aux variables partagées disponibles aux serveurs.

Ces références peuvent être soit des requêtes d'écriture, soit des requêtes de lecture.

Initialement, chaque processeur effectue des calculs. Après un intervalle de temps que nous appellerons temps d'interarrivée des requêtes t_{arr} il réclame un objet à sa mémoire locale. Ce temps est une variable aléatoire ayant une distribution arbitraire de moyenne T_{arr} unités de temps (UT) et dépend du types d'instruction. Le temps d'accès à une mémoire locale t_{mem} est une variable aléatoire qui est directement proportionnelle à la taille de l'objet. Cette variable aléatoire est distribuée arbitrairement. Sa moyenne est de T_{mem} UT. Si l'objet est absent localement, alors le chargement se fera à partir du serveur par l'intermédiaire du réseau de communication. Le temps de réponse à la mémoire globale t_{ser} est une variable aléatoire distribuée arbitrairement de moyenne T_{ser} UT. Le réseau de communication met un nombre arbitraire de temps pour transférer un objet. Ce temps de transmission t_{res} suit également une distribution arbitraire de moyenne T_{res} UT. D'après l'hypothèse 13, nous supposons que toutes le distributions arbitraires mentionnées ci-haut sont exponentielles.

3- Quantification des paramètres.

Les objets du système sont de deux types: privé si l'objet n'est accessible que par un seul poste ou partagé si l'objet est accessible par tout le monde. Pour notre analyse, nous considérons que tous les objets du système sont partagés. Autrement dit, le temps d'accès à ces objets est compris dans le temps d'interarrivée des requêtes T_{arr}. Nous avons au total B_p blocs partagés dans le système et chaque bloc ayant une taille de une unité mémoire. Autrement dit, un objet occupe une place mémoire. Cette unité est la même aux serveurs et aux postes. D'après l'hypothèse 10, chaque serveur dispose de suffisamment de mémoire pour emmagasiner tous les objets partagés du système. Par contre, d'après l'hypothèse 5, chaque poste a une mémoire locale de capacité C_{mem} inférieure à celui nécessaire pour emmagasiner les B_p objets; ce qui veut dire que pour charger un objet à partir d'un autre poste, il faut faire de la place si jamais l'espace disponible est insuffisant. Cette opération se fait en exécutant localement un algorithme de remplacement. Dans plusieurs systèmes, c'est l'algorithme LRU (Least Recently Used) qui est utilisé [Tanenbaum 87]. Pour simplifier notre analyse, nous utiliserons un algorithme de remplacement où le choix du bloc se fait par un tirage aléatoire. Si le poste de travail est propriétaire du bloc choisi, alors le serveur redevient propriétaire selon notre protocole de mise à jour. Une requête issue d'un processeur fait référence à un des objets partagés avec une probabilité uniforme égale à $q_j = (1/B_p)$ selon l'hypothèse 7 d'uniformité de la chaîne de référence. Cette requête est une écriture avec une probabilité de W et une lecture avec probabilité de 1-W. Selon l'hypothèse 9, chaque donnée est associée à un serveur principal en particulier, tous les serveurs principaux ont la même probabilité d'être adressés par une requête issue d'un poste de travail.

Un processeur du système peut être dans l'un des états suivants: actif ou en attente. Un processeur est dit actif lorsqu'il effectue des calculs. Il est dit en attente à partir du moment où il émet une requête jusqu'à l'instant où il obtient une réponse à celle-ci. Si la requête peut être satisfaite localement, alors le processeur redevient actif immédiatement après que la requête soit terminée. Par contre, si la requête ne peut pas être satisfaite localement, par exemple lors d'un chargement d'un bloc depuis un des serveurs ou lors d'une demande du privilège d'écriture ou encore lors de la mise à jour des autres copies du système, alors il va attendre un délais supplémentaire égal au temps de transfert de priviège, ou de mise à jour de copies pour redevenir actif. La proportion du temps où le processeur demeure actif définit son taux d'utilisation que nous désignons par ρ_{proc}. Si nous désignons par T_{rep} le temps de réponse moyen du système qui est une quantité finie d'après l'hypothèse 4, alors l'expression du taux d'utilisation du processeur est donnée par la formule:

$$\rho_{proc} = \frac{T_{arr}}{T_{arr} + T_{rep}}. \qquad (4.0)$$

Puisque le temps de réponse moyen du système dépend directement des activités des processeurs et de leur nombre, alors nous désignons la puissance de calcul (P_{cal}) qui est donné par

$$P_{cal} = N \times \rho_{proc} \qquad (4.1)$$

et qui est le produit du nombre de postes, N, par le taux d'utilisation moyen des processeurs comme paramètres caractéristiques de notre analyse [Bhuyan 88].

4.2.1.2 Le modèle

Le système décrit dans les sections antérieures peut être représenté par le modèle de la figure 4.1.

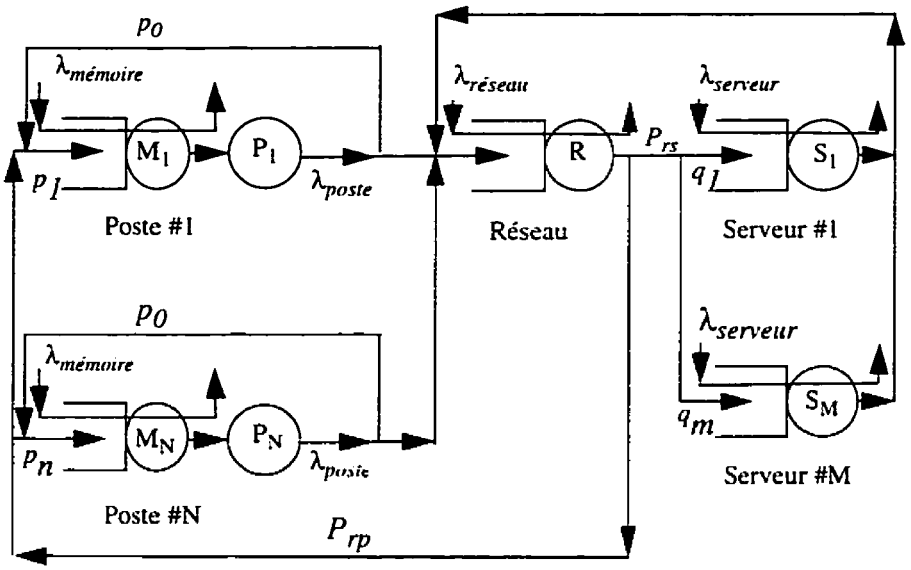


Figure 4.1 Modèle du processeur

Il s'agit d'un réseau de files d'attente constitué de chaînes de routage ouvertes et fermées similaire à celui développé par Bhuyan [Bhuyan 88]. Les processeurs sont représentés des files d'attente à capacités infinies ou centres de délai P_i , $i = 1, 2, .., N$ où N est le nombre de postes dans le système. Ceci est dû au fait qu'il n'existe d'interaction entre les processeurs. Les mémoires locales sont représentées par des files d'attentes PAPS (premier arrivé premier servi) M_i , $i = 1, 2, .., N$. Cette représentation s'explique par le fait que les mémoires locales peuvent être sollicitées par les autres processeurs, par exemple pour un transfert de privilège, une mise à jour ou encore une invalidation. Les SP sont représentés par des files d'attente PAPS S_j , $j = 1, 2, .., M$ où M est le nombre de SP dans le système. Ceci est dû au fait qu'ils sont partagés par tous les processeurs. Le réseau est également une file d'attente PAPS R car il est également partagé. Les clients sont les activités des processeurs qui peuvent être des calculs, ou des opérations de lecture et d'écriture. Il y a un total de N clients chacun appartenant à sa propre chaîne de routage

fermée car d'après notre protocole il n'y a pas d'interaction entre les processeurs. Autrement dit, un client provenant d'un processeur ne peut pas visiter un autre poste que celui auquel il appartient. Chaque client est donc associé à un processeur.

Les chaînes ouvertes $\lambda_{mémoire}$, $\lambda_{serveur}$ et $\lambda_{réseau}$ représentent les débits des requêtes autres que les requêtes de lecture et d'écriture. Nous les appelerons tâches de fond du système. Ces tâches sont par exemple la mise à jour des blocs, le transfert de privilège pour ne citer que celles là. Elles seront décrites dans une section à venir.

Trajet des clients

Un client est initialement dans le processeur représenté par le centre de délai P_i pendant un intervalle de temps aléatoire de moyenne, T_{arr}, représentant le processeur en activité. Puis lorsque ce temps est écoulé, le processeur réclame un bloc en formulant une requête. Si la requête peut être satisfaite localement, alors elle est envoyée à la file d'attente de la mémoire locale qui lui est associée. Elle y passe un certain temps T_{mem} égal au temps de transfert du bloc et repart par la suite à son point d'origine. Si par contre la requête ne peut pas être satisfaite localement, alors elle rejoint la file d'attente représentant le réseau. Lorsqu'elle obtient le réseau elle l'utilise pendant un temps T_{res} égal à son temps moyen de transmission , puis est transférée dans la file d'attente du serveur. Après l'avoir utilisé, la requête se rend dans la file d'attente du réseau. Par la suite, elle rejoint la file d'attente de sa mémoire locale. De cette file, elle retourne dans son centre de délai et le cycle recommence.

Il faut aussi noter que, selon notre protocole de mise à jour en écriture, une requête peut en

générer plusieurs autres. Par exemple si on veut modifier un bloc et que celui-ci est dans l'état RO, alors il faut aller obtenir le privilège d'écriture, modifier le bloc et faire la mise à jour de toutes les autres copies du système, ou bien, si on modifie un bloc qui était préalablement dans l'état RW, alors il faut mettre à jour toutes les autres copies du système. On constate que l'effet de ces transactions sur le système est d'engorger les files par où elles transitent. Ainsi, celles-ci seront modélisées par des chaînes ouvertes ayant des taux d'arrivée et de départ identiques. Ces différentes tâches de fond sont les suivantes:

- obtention du privilège d'écriture de taux λ_{Obt_Priv};
- mise à jour des copies dans les mémoires locales par les serveurs λ_{Serv_Post};
- mise à jour des SP par les postes λ_{Post_Serv};
- mise à jour des SS par les SP λ_{Serv_Serv};
- remplacement des blocs λ_{Remp}.

Cependant, l'utilisation du réseau par certaines de ces transactions dépend du mécanisme utilisé pour propager l'information. Par exemple, si la diffusion est utilisée pour faire la mise à jour, chaque poste ayant une copie valide de ce bloc obtient une seule requête, alors que le réseau est peturbé par toutes les requêtes en même temps. Puisque nous connaîssons le trajet de chacune de ces transactions, nous pouvons calculer le débit total traversant chaque file. Pour le serveur, le réseau et la mémoire, nous désignons ces quantités respectivement par $\lambda_{serveur}$, $\lambda_{réseau}$ et $\lambda_{mémoire}$, qui apparaissent sur la figure 4.1. Vu l'importance de leurs effets sur le système, nous consacrerons plus loin une section au calcul des débits de ces transactions.

La résolution de ce modèle nous donne le temps de réponse moyen T_{rep} qui nous permet de caluler ρ_{proc} et d'en déduire le débit λ_{poste} traversant le processeur par la formule

$$\lambda_{poste} = \frac{\rho_{proc}}{T_{arr}} \tag{4.2}$$

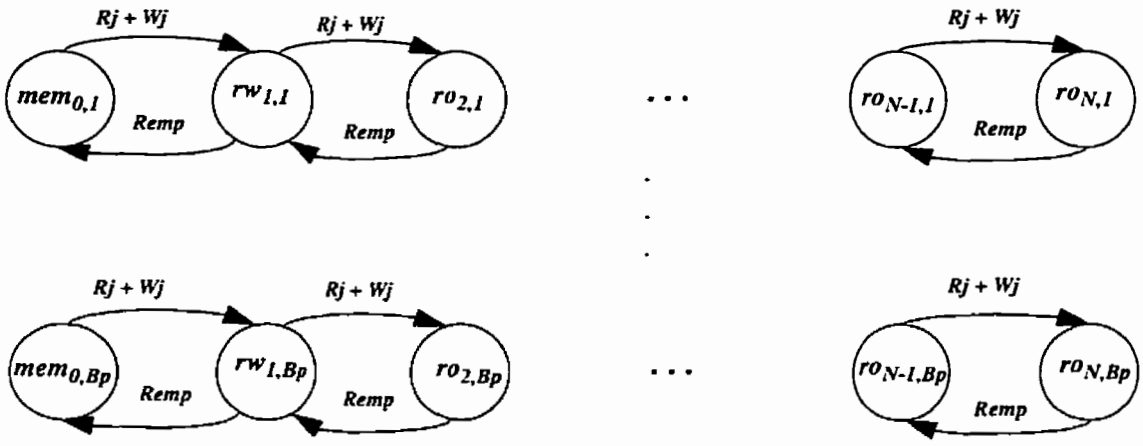
Pour le résoudre, il faut d'une part quantifier l'évolution des clients par leurs probabilités de routage, et d'autre part déterminer les taux des chaînes ouvertes. Ces quantités dépendent non seulement de l'état des blocs mais également du nombre d'exemplaire de ceux-ci. Par exemple le trajet d'une requête générée par un processeur dépend de l'état du bloc dans sa mémoire. Ou bien, une opération de mise à jour d'un bloc par le serveur principal est nécessaire si et seulement si le bloc est partagé par un autre poste. L'effet de cette opération sur les performances du système seront plus important lorsque le nombre de copies à mettre à jour sera grand. Ainsi pour calculer son taux, il faut avoir une distribution du nombre de copie dans le système. Ainsi il faut trouver un modèle des données qui représente d'une part le changement dynamique de l'état des bloc et de l'autre l'évolution du nombre de copies des bloc dans le système.

4.2.2 Modèle des données

Dans ce modèle, il s'agit de modéliser le comportement des données dans le système. En utilisant l'hypothèse 7 d'uniformité de la chaîne de référence et l'hypothèse 6 d'indépendance des données, nous allons analyser un seul bloc. Le modèle obtenu sera identique pour les autres. Le processeur agit sur un bloc de plusieurs façons: soit pour obtenir une copie, soit pour la modifier ou soit la remplacer. Ces différentes actions ont deux influences différentes sur le modèle. D'une part, le nombre de copies varie, et d'autre part, l'état de la donnée dans une mémoire locale change. Nous allons analyser à tour de rôle ces phénomènes.

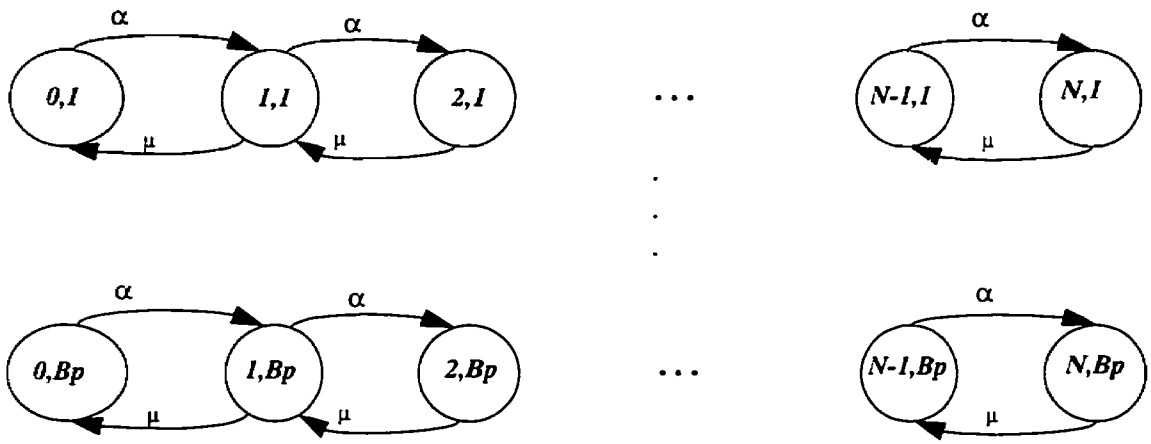
1 - Analyse du nombre de copies

À chaque fois que survient une faute d'objet, il faut aller au serveur. Dans ce cas, le nombre de copies augmente. Cependant, lorsqu'un bloc est choisi pour être remplacé par un algorithme de remplacement, alors le nombre de copies diminue. Ainsi l'évolution du nombre de copie dans le système peut être représenté par le diagramme de transition du nombre de copies des blocs de la figure 4.2.



$R_j + W_j$: Chargement d'un bloc absent lors d'une requête de lecture ou d'écriture
Remp: Remplacement d'un bloc

Figure 4.2 Diagramme de transition du nombre de copies des blocs dans le système.



α: Taux de chargement d'un bloc absent lors d'une requête de lecture ou d'écriture
μ: Taux de remplacement d'un bloc

Figure 4.3 Chaîne de Markov du nombre de copies des blocs dans le système.

Ce diagramme modélise toutes les données partagées du système. Pour représenter un état de la chaîne, nous utilisons un nom indicé de deux nombres qui sont respectivement le nombre de copies et le numéro du bloc. Le nom de l'état est représentatif de l'état du bloc. Si le bloc est disponible aux serveurs uniquement, alors l'état est *mem*. Si un poste possède une copie du bloc et est le propriétaire, alors l'état est *rw*. Autrement, l'état est *ro*. Une transitions vers la droite s'effectuent à la fin d'une requête de lecture ou d'écriture par un poste n'ayant pas une copie du bloc. Cette condition de transition est notée $R_j + W_j$. Une transition vers la gauche s'effectue à la suite d'un remplacement de bloc. Cette condition de transition est notée P_g sur la figure. Selon l'hypothèse 13, nous pouvons remplacer les conditions de transition de ce diagramme par leurs taux respectifs, et obtenir la chaîne de Markov du nombre de copies des blocs de la figure 4.3. Cette chaîne est à temps continu et à états discrets. Il faut noter que l'instant de transition est la fin d'une requête. De cette chaîne, on obtient les probabilités stationnaires $p_j[i]$ d'avoir i copies du

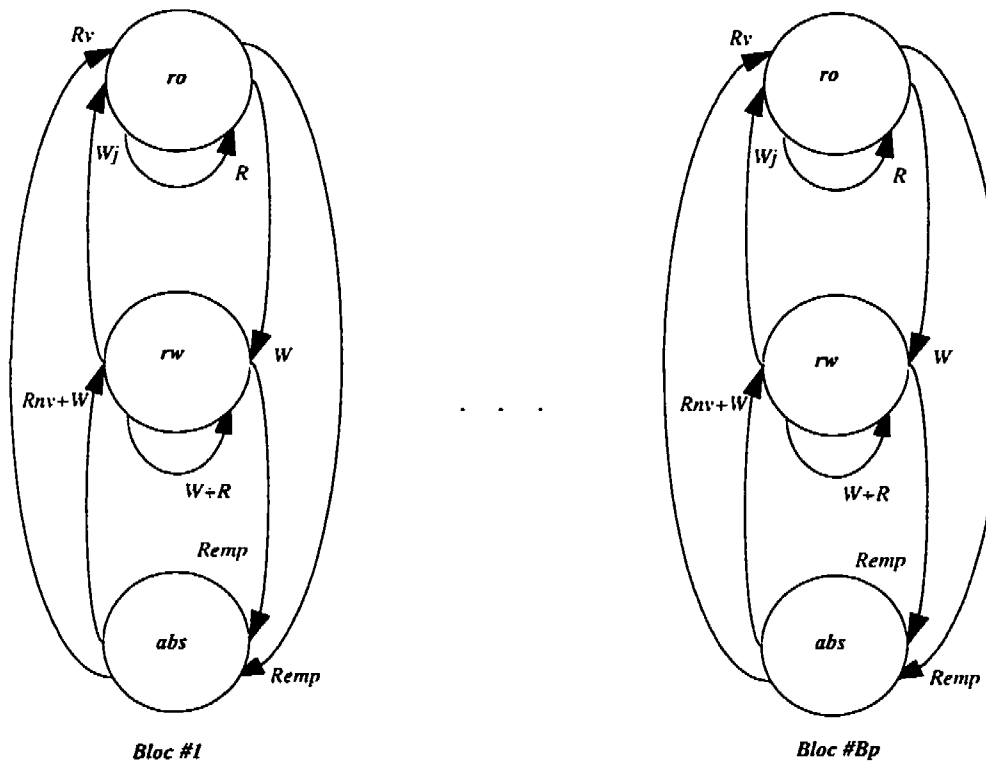
bloc *j* dans le système.

Cependant, pour la résoudre, il faut donc connaître le débit des fautes de blocs et leur taux de remplacement. Or chacune de ces quantités représentent une proportion du débit λ_{poste} traversant chaque processeur, proportion qui est déterminer par la distribution des états des blocs dans les mémoires. Par exemple, dans le cas d'une faute de bloc, cette proportion est la probabilité que le bloc soit dans l'état absent. Cette probabilité peut être obtenue par le modèle qui suit.

2 - Analyse des états des copies

D'après notre protocole de mise à jour en écriture, le changement d'état d'un bloc dépend de son état dans la mémoire locale des autres postes, des opérations de lecture et d'écriture et de l'algorithme de remplacement. Ce changement peut être représenté par le diagramme de transitions d'état des blocs relatifs à un poste spécifique de la figure 4.4 que nous avons déjà rencontré au chapitre précédent.

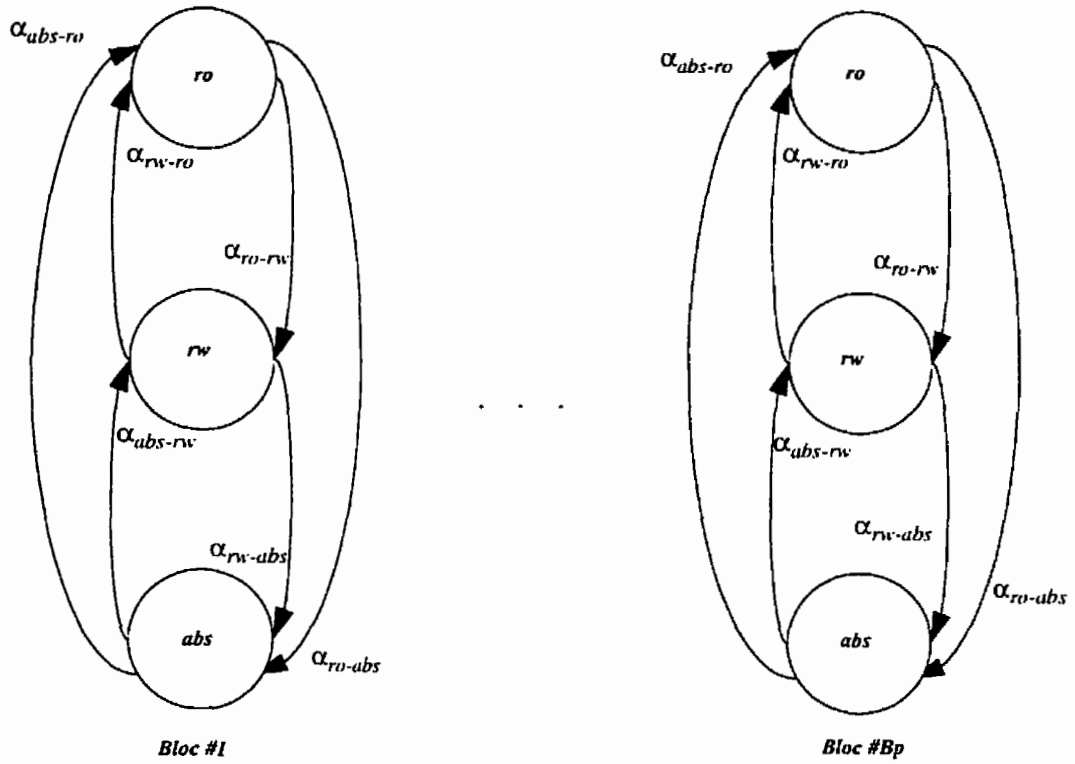
Ce diagramme modélise toutes les transitions possibles d'un bloc dans une mémoire locale ainsi que les conditions de transition. De l'état *abs*, on peut transiter vers l'état *rw* s'il s'agit du premier accès au bloc ou vers l'état *ro* dans le cas contraire. La transition inverse survient lorsque le bloc est choisi par un algorithme de remplacement. De l'état *rw*, on peut transiter vers l'état *ro* si un autre poste effectue une opération d'écriture. La transition inverse se réalise lorsque le poste ayant un bloc dont l'état est *ro* demande à le modifier.



W : écriture
Wj: écriture par un autre poste
R : lecture
Rv : lecture et copie valide à un autre poste
Rnv: lecture et pas de copies valides ailleurs
Remp: remplacement

Figure 4.4 Diagramme de transitions des états des blocs en mémoire locale

Selon l'hytothèse 13, nous pouvons remplacer les conditions de transition de ce diagramme, par leurs taux respectifs et obtenir la chaîne de Markov de transition d'états des blocs en mémoire locale de la figure 4.5.



$\alpha_{x\text{-}y}$: Taux de transition de l'état x vers l'état y

Figure 4.5 Chaîne de Markov des états des blocs en mémoire locale

Comme dans le cas précédant, les transitions se font à la fin d'une requête et les taux de transitions représentent des proportions du débit λ_{poste} traversant le processeur. Certains de ces taux dépendent aussi du nombre d'exemplaire d'une donnée dans le système. Par exemple de l'état *abs* on peut transiter à l'état *ro* ou à l'état *rw* après une requête de lecture. S'il s'agit de la seule copie dans le système, alors le nouvel état est *rw*; sinon, il est *ro*. Ainsi, pour connaitre l'état final véritable, il faut connaître la distribution du nombre de copies dans le système. Ce modèle a été obtenu précédamment. Cette chaîne nous donne les probabilités stationnaires $\pi_j[abs]$, $\pi_j[rw]$, $\pi_j[ro]$ que le bloc j soit dans chacun des états *abs*, *rw* et *ro*.

4.2.3 Solutions du modèle

Le modèle analytique est constitué du modèle du processeur et des deux chaînes de Markov qui constituent le modèle de données.

Pour résoudre le modèle du processeur, il faut au préalable exprimer la dynamique du système. Ce qui est matérialisable par le calcul des probabilités de routage ainsi que les débits des chaînes ouvertes. Ces paramètres peuvent être calculés en utilisant les résultats obtenus en résolvant les deux chaînes de Markov qui constituent le modèle des données. Considérons la chaîne de Markov qui représente le nombre de copie. Pour la résoudre, il faut déterminer les taux de transition. Or ces taux peuvent être calculés si on connaît le débit λ_{poste} traversant chaque processeur dans le modèle du processeur, et si on résout la chaîne de Markov représentant l'état des blocs. Dans la chaîne de Markov représentant l'état des blocs, les taux de transition ne peuvent être déterminé que si on connaît le débit λ_{poste} traversant chaque processeur dans le modèle du processeur, et les probabilités d'être dans chacun des états de la chaîne de Markov qui représente le nombre de copie. On voit donc qu'il existe des interactions entre les modèles. Ces interactions sont résumées à la figure 4.6.

Le modèle du processeur fournit λ_{poste} comme paramètre aux deux autres modèles et obtient d'eux les débits des chaînes ouvertes et p_0. Les deux chaînes de Markov s'échangent leurs probabilités stationnaires et les utilisent pour calculer leurs taux de transition respectifs. Pour résoudre notre modèle, il est donc impératif de passer par une étape d'initialisation. Ensuite, il faut itérer sur les différents modèles jusqu'à convergence.

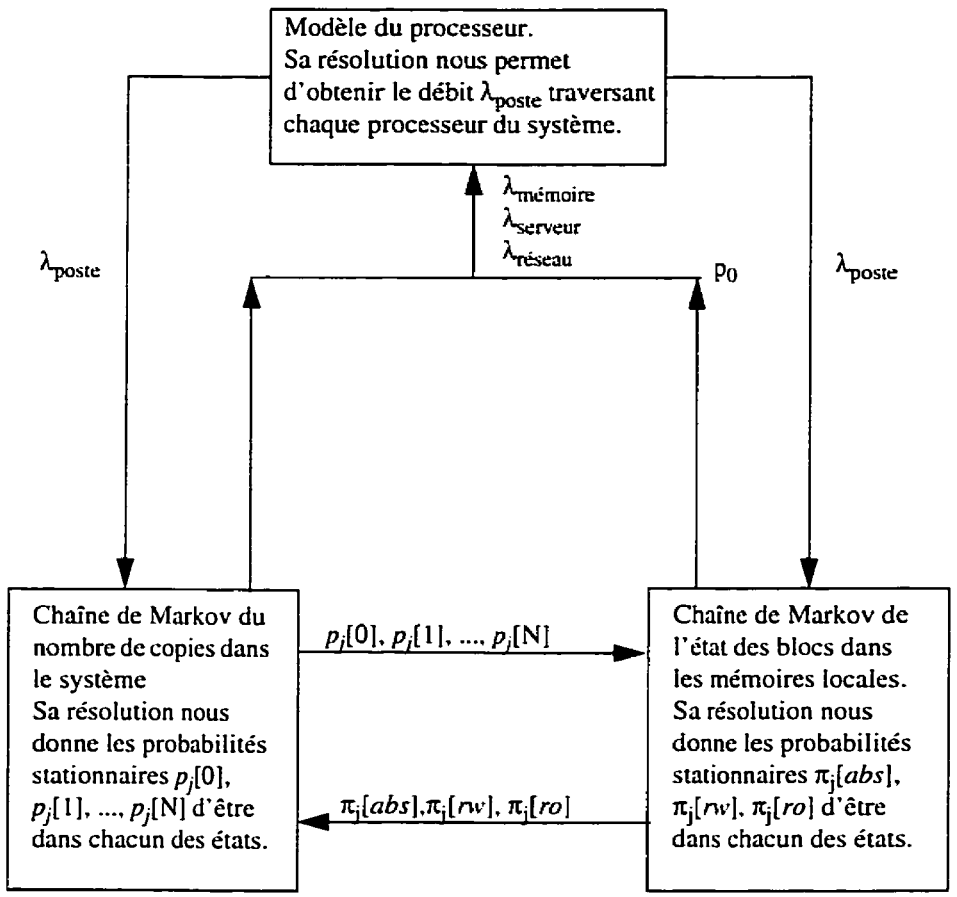


Figure 4.6 Interactions entre les modèles

Cependant, cette solution du modèle original n'est pas pratique. Lorsque la taille du système est grande, l'algorithme de la valeur moyenne (MVA) utilisé pour le calcul du temps de réponse dans le modèle du processeur exige $F \times \prod_{c=1}^{C} (N_c + 1)$ d'espace mémoire pour sauvegarder les données du programme [Bruel 80]. Dans cette équation, F désigne le nombre total de files, C le nombre total de classes et N_c le nombre de clients pour la classe c. Une solution améliorée peut être obtenu en utilisant les hypothèses du modèle .

On peut constater que les clients dans la file d'attente de la mémoire locale des postes appartiennent tous à des chaînes ouvertes sauf un client qui appartient à une chaîne

fermée. Si on utilise l'approximation de Reiser selon laquelle l'effet des chaîne ouvertes sur la file d'attente est d'augmenter le temps de service [Reiser 79], alors cette file peut être remplacée par un centre de délai comme le processeur. On peut avec ce modèle réduire le nombre de chaîne de routage à une et augmenter le nombre de clients à N, où N représente la somme des clients de chacune des anciennes chaînes. Ce modèle est représenté à la figure 4.7. Les files d'attente à capacités infinies tiennent compte du fait que les clients ne doivent pas attendre. L'avantage de ce modèle à une seule chaîne est qu'il ne demande plus que $F \times (N+1)$ espace mémoire pour le tableau du temps de réponse des files.

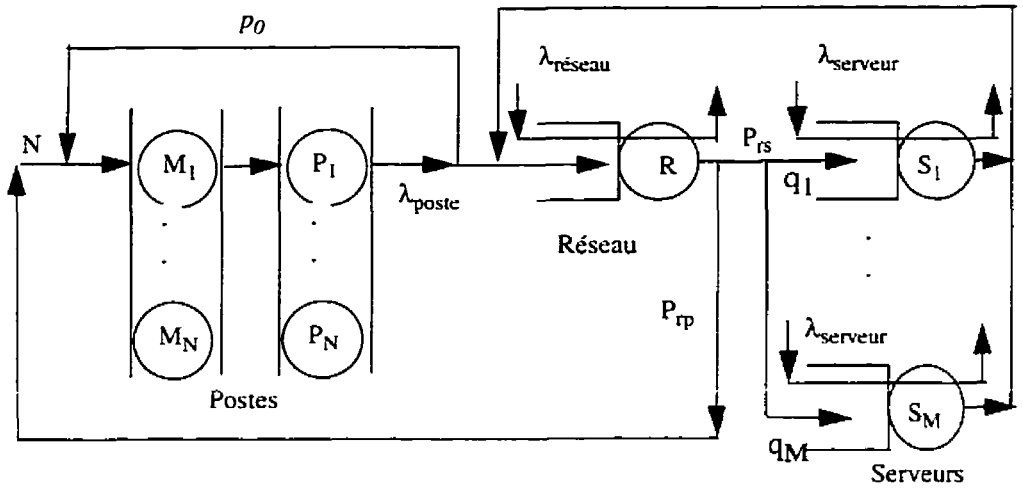


Figure 4.7 Modèle du processeur avec une chaîne de routage et N clients

Calcul des taux de transition

Dans les formules ci-dessous, λ_{poste} est le débit du système mesuré au proceseur. Le terme q_j est la probabilité uniforme de choisir un bloc partagé d'indice j et est égal à B_p^{-1} où B_p est le nombre total de bloc dans le système. C_{mem} est la capacité de chaque mémoire locale, $Cmem^{-1}$ est la probabilité de choisir un bloc lorsque la mémoire est pleine. W est la

probabilité qu'une requête soit une demande écriture et *1-W* la probabilité qu'elle soit une demande de lecture. Lorsque $C_{mem} <= B_p$, le terme $(1 - C_{mem}/B_p)$ est la probabilité qu'un bloc partagé au moins soit absent d'une mémoire locale.

De la chaîne de Markov du nombre de copies des blocs de la figure 4.3, nous obtenons pour un seul bloc, la chaîne de Markov représentant le nombre de copies du bloc de la figure 4.8.

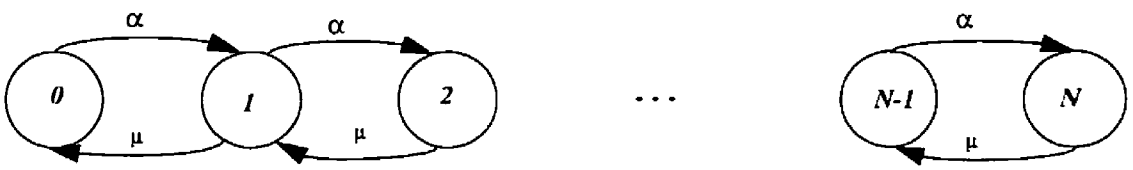


Figure 4.8 Chaîne de Markov du nombre de copies d'un bloc.

Le taux α est le débit des fautes de page. Ce taux est donné par l'expression

$$\alpha = \lambda_{poste} \times q_j \times \pi_j[abs] \qquad (4.3)$$

où $\pi_j[abs]$ est la probabilité que le bloc *j* soit absent dans la mémoire du poste considéré.

Le taux μ est le débit de remplacement des blocs et est donné par la formule

$$\mu = \lambda_{poste} \times (\pi_j[rw] + \pi_j[ro]) \times (\frac{1}{C_{mem}}) \times \left(1 - \frac{C_{mem}}{B_p}\right) \times q_j. \qquad (4.4)$$

Dans cette formule les deux premiers termes représentent le débit des fautes générées par les blocs absents de la mémoire.

Cette chaîne est apériodique et irréductible et peut être résolue directement à partir des équations de Chapman-Kolmogorov[Kleinrock 75]. De ces équations, on obtient les

probabilités stationnaires $p[i]$ d'avoir i copies d'une donnée dans le système.

La chaîne de Markov représentant les états d'un bloc dans une mémoire locale est représenté à la figure 4.9.

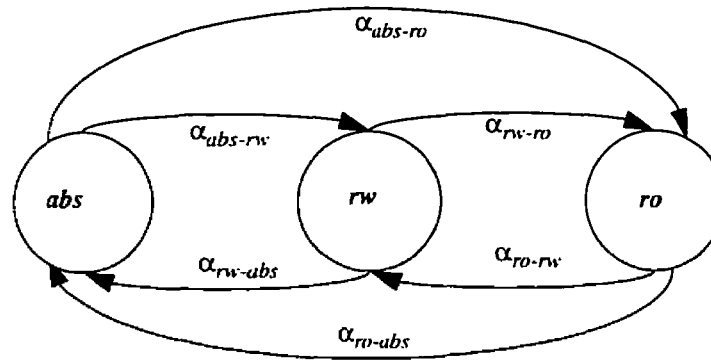


Figure 4.9 Chaîne de Markov pour le changement d'état d'un bloc en mémoire locale

Le calcul des taux de transition est l'objet de la section ci-dessous.

Transition de *abs* à *rw*

Cette transition survient après qu'une opération de lecture adressée à une donnée disponible uniquement qu'aux serveurs, soit terminée, ou encore après une requête d'écriture. Le taux de transition est donné par la formule

$$\alpha_{abs-rw} = \lambda_{poste} \times q_j \times (1 - W) \times p[0] + \lambda_{poste} \times q_j \times W \qquad (4.5)$$

où $p[0]$ est la probabilité que le bloc soit disponible au serveur uniquement.

Transition de *abs* à *ro*

Cette transition survient a la fin d'une requête de lecture sur une donnée disponible aux autres postes. Le taux de transition est donné par

$$\alpha_{abs-ro} = \lambda_{poste} \times q_j \times (1 - W) \times (1 - p[0]) \quad , \tag{4.6}$$

où le facteur (1-p[0]) est la probabilité que le bloc soit disponible dans un autre poste.

Transition de *rw* à *abs* et de *ro* à *abs*

Cette transition survient lorsque le bloc en mémoire locale est choisi pour être remplacé. Les taux de transition respectif sont donnés par

$$\alpha_{rw-abs} = \lambda_{poste} \times (\frac{1}{C_{mem}}) \times q_j \tag{4.7}$$

et

$$\alpha_{ro-abs} = \lambda_{poste} \times (\frac{1}{C_{mem}}) \times q_j \tag{4.8}$$

qui représentent le produit du débit du poste par la probabilité uniforme qu'un bloc présent en mémoire soit choisi pour être remplacé par la probabilité que le bloc considéré soit choisi.

Transition de *ro* à *rw*

Cette transition est completée à la fin d'une requête d'écriture. Le taux est:

$$\alpha_{ro-rw} = \lambda_{poste} \times q_j \times W \tag{4.9}$$

Transition de *rw* à *ro*

Pour qu'un bloc perde le privilège d'écriture, il faut qu'un autre poste veuille le modifier. Le taux est

$$\alpha_{rw-ro} = (N - 1) \times \lambda_{poste} \times q_j \times W \tag{4.10}$$

Le terme *(N -1)* est le nombre de postes voulant modifier le bloc.

La chaîne de Markov de la figure 9 a les mêmes caractéristiques que la précédante et peut être résolue de la même manière. On obtient les probabilités stationnaires $\pi_j[abs]$, $\pi_j[rw]$ et $\pi_j[ro]$ d'avoir une copie du bloc j localement dans l'un des états *abs, rw et ro* respectivement.

4.1.3.1 Probalilités de routages

La dynamique du système se traduit par le calcul des probabilités de routage. Avec l'obtention des probabilités stationnaires précédentes, nous sommes prêt à calculer les probabilités de routage p_0, P_{rp}, P_{rs}, p_j où $j = 1, 2....N$, N étant le nombre de postes et q_k, $k = 1, 2, ..., M$, avec M étant le nombre de serveurs principaux.

Pour qu'une requête soit satisfaite localement, il faut que la donnée soit d'abord présente. Si une requête de lecture est faite pour obtenir un bloc localement dans l'état *rw* ou *ro*, il ne peut jamais survenir de faute. C'est également le cas si une requête d'écriture est faite pour obtenir un bloc localement dans l'état *rw*. La probabilité p_0 qu'une requête soit satisfaite localement se calcule par l'expression

$$p_0 = \pi_j[rw] + \pi_j[ro] \times (1 - W) \qquad (4.11)$$

qui est la probabilité d'être dans l'état *rw* plus la probablité d'être dans l'état *ro* et faire une demande de lecture. La probabilité qu'une requête utilise le réseau pour être satisfaite par le SP est $1\text{-}p_0$. La probabilité qu'une requête sortant du réseau se dirige vers le serveur est $P_{rs} = 1/2$. La probabilité qu'elle se dirige vers un SP en particulier est q_k x P_{rs} où $q_k = 1/M$, $k = 1, 2, ..., M$. La probabilité qu'elle se dirige vers les postes est $P_{rp} = 1\text{-}P_{rs} = 1/2$. La probabilité qu'elle se dirige vers un poste en particulier est p_j x P_{rp} où $p_j = 1/N$, $j = 1, 2, ..., N$.

4.2.3.2 Débit des chaînes ouvertes

Le debit de chaque chaîne ouverte s'obtient de la manière suivante:

1- Remplacement des blocs

Un remplacement de bloc survient lorsque dans une mémoire locale, un bloc est choisi par l'algorithme de remplacement, alors il faut aller informer le serveur principal. Ces transactions sont donc générées par un poste et impliquent le réseau et le serveur responsable.

Le taux de ces transactions peut s'obtenir par l'expression suivante:

$$\lambda_{Remp} = N \times \lambda_{poste} \times \pi_j[abs] \times (\frac{1}{C_{mem}}). \qquad (4.12)$$

2 - Mise à jour du serveur par un poste.

Cette opération survient après une opération d'écriture. Il faut donc mettre à jour la donnée au serveur. Cette transaction est donc générée par un poste qui génère la requête d'écriture et implique le réseau et le serveur de la donnée en question.

Le taux de ces transactions peut être calculé par l'expression:

$$\lambda_{Post-Serv} = N \times \lambda_{poste} \times W \qquad (4.13)$$

qui est le débit total des postes multiplié par le taux d'écriture.

3- Mise à jour des blocs dans les mémoire locales par le serveur

Il s'agit de la suite de la transaction précédente. Le serveur est chargé de mettre à jour toutes les données disponibles. Cette transaction générée par le serveur du bloc implique le réseau et les mémoires locales des postes qui ont une copie du bloc. Etant donné que le nombre de copies peut être grand cette opération peut se réaliser en utilisant deux techniques: la diffusion ou le chaînage.

Le taux de ces transactions peut être calculé par l'expression:

$$\lambda_{Serv-Post} = \lambda_{poste} \times W \times \pi_j[ro] \qquad (4.14)$$

qui est le débit du système par le taux d'écriture par la probabilité qu'il y ait au moins une copie à un autre poste.

4- Obtention de privilège

Cette transaction survient lorsque pour une requête d'écriture, la donnée locale ne se trouve pas dans l'état RW et que le serveur n'est pas propriétaire de la donnée. Elle est initiée par le serveur et affecte le réseau et le poste détenant le privilège. Le débit de ces transactions est

$$\lambda_{Obt-Priv} = \lambda_{poste} \times (1 - \pi_j[rw]) \times W \times \pi_j[rw], \qquad (4.15)$$

qui est le produit du débit du système par la probabilité que l'état du bloc ne soit pas RW et que cette donnée soit disponible à un autre poste.

5 - Mise à jour des serveurs secondaires par le serveur principal

Il faut effectuer cette opération chaque fois qu'il ya une transaction qui change un bloc ou

qui modifie sa localité. Elle est donc initiée par le serveur principal et implique tous les autres serveurs. Cette opération peut s'effectuer de deux façons: soit par chaînage ou par diffusion.

Le débit de ces transactions est

$$\lambda_{Serv-serv} = N \times \lambda_{poste} \times (1 - \pi[rw] - \pi[ro]) \times (1 - w) + \lambda_{w}, \qquad (4.16)$$

où

$$\lambda_{w} = N \times \lambda_{poste} \times W . \qquad (4.17)$$

La première expression tient compte juste des fautes d'objet, et la deuxième du fait qu'il faut modifier un bloc physique présent.

Dans le cas de la diffusion, le taux traversant le réseau va être multiplié par le nombre de serveurs à mettre à jour pour tenir compte du fait que cette opération se fait en un seul coup par le serveur principal sur le réseau.

6 - Bilan des débits des chaînes ouvertes à travers les files

a) Mémoire

Pour la mémoire le bilan est

$$\lambda_{memoire} = \lambda_{Obt-priv} + \lambda_{Post-serv}. \qquad (4.18)$$

b) Serveur principal

Pour un SP, le bilan est donné par l'expression:

$$\lambda_{serveur} = \frac{(\lambda_{Remp} + \lambda_{Post-serv} + SS \times \lambda_{Serv-serv})}{S}, \qquad (4.19)$$

ou SS est le nombre de serveurs secondaires.

c) Réseau

Dans le cas d'une diffusion simulée, le débit traversant le réseau est donné par

$$\lambda_{reseau} = 2 \times (\lambda_{Remp} + \lambda_{Post-serv}) + SS \times \lambda_{Serv-serv} + \lambda_{ds}, \quad (4.20)$$

où le dernier terme est donné par l'expression

$$\lambda_{ds} = N \times N_{copies} \times \lambda_{Serv-post} + N \times \lambda_{Obt-priv},$$

où $N_{copies} = \sum_{i=2}^{N} (i-1) \times p[i]$.

Dans le cas d'une diffusion véritable, le débit traversant le réseau est donné par

$$\lambda_{reseau} = 2 \times (\lambda_{Remp} + \lambda_{Post-serv}) + \lambda_{Serv-serv} + \lambda_{dv}. \quad (4.21)$$

où le dernier terme est donné par l'expression

$$\lambda_{dv} = N \times \lambda_{Serv-post} + N \times \lambda_{Obt-priv}. \quad (4.22)$$

4.2.3.3 Algorithme

Si on prend comme hypothèse que dans le modèle du processeur, le temps de service de chaque file suit une distribution exponentielle, alors le système forme un réseau à forme produit [Bruel 80]. La solution de ce modèle peut être obtenu en utilisant l'algorithme MVA sous sa forme exacte [Bruel 80]. L'obtention des paramètres du modèle se fait selon l'algorithme suivant:

Paramètres d'entrée:

- N, M qui sont respectivement le nombre de processeur et de serveurs,
- B_p qui est le nombre total de blocs partagés,
- W, T_{arr} qui désignent respectivement le taux d'écriture et le temps d'interarrivée des requêtes. Ces quantités dépendent du programme.
- $T_{rés}$, T_{mem}, T_{ser}, C_{mem}, qui sont respectivement le temps de transmission du réseau, le temps de réponse de la mémoire, le temps de transfert du serveur et la capacité d'une mémoire locale. Ces quantités sont plutôt des paramètres architecturaux.

1 - Initialiser l'utilisation du processeur $\rho_{proc} = 1$ et le critère de convergence;

2 - Résoudre la chaîne de Markov du nombre de copie des blocs;

3 - Résoudre la chaîne de Markov du changement d'état des blocs;

4- Calculer les probabilités de routage p_0;

3 - Calculer les débits des chaînes ouvertes $\lambda_{serveur}$, $\lambda_{mémoire}$ et $\lambda_{réseau}$;

4 - Calculer l'utilisation ρ_{proc} du processeur à partir du modèle du processeur;

5 - Si ρ_{proc} ne converge pas, alors aller à l'étappe 2.

4.3 Le modèle à invalidation

Il s'agit d'obtenir une représentation analytique de ce modèle à partir de sa description physique et de son fonctionnement logique.

4.3.1 Modèle du processeur

La modélisation de ce modèle peut être faite par analogie avec celui à mise à jour. En effet, les deux modèles physiques sont identiques. Cependant, leurs fonctionnements logiques sont différents. Cette différence au niveau du protocole de cohérence survient lorsqu'il faut compléter une requête d'écriture. Dans le cas du modèle de mise à jour, il faut mettre à jour toutes les autres copies disponibles, alors que pour le modèle d'invalidation en écriture, il faut les invalider. Ces opérations sont modélisées par des chaînes ouvertes qui ralentissent les ressources qu'elles utilisent. Malgré que les opérations d'invalidation que de mise à jour soient de natures différentes, elles affectent les mêmes ressources. Puisque les activités principales des processeurs sont les mêmes dans les deux modèles (à savoir calculs, lectures, écritures), elles peuvent être représentées par le même modèle du processeur. La différence entre ces deux modèles est donc

quantitative. Elle va se manifester sur les valeurs des probabilités de routage ainsi que des débits des chaînes ouvertes car ces quantités dépendent du nombre et de l'état des copies de blocs.

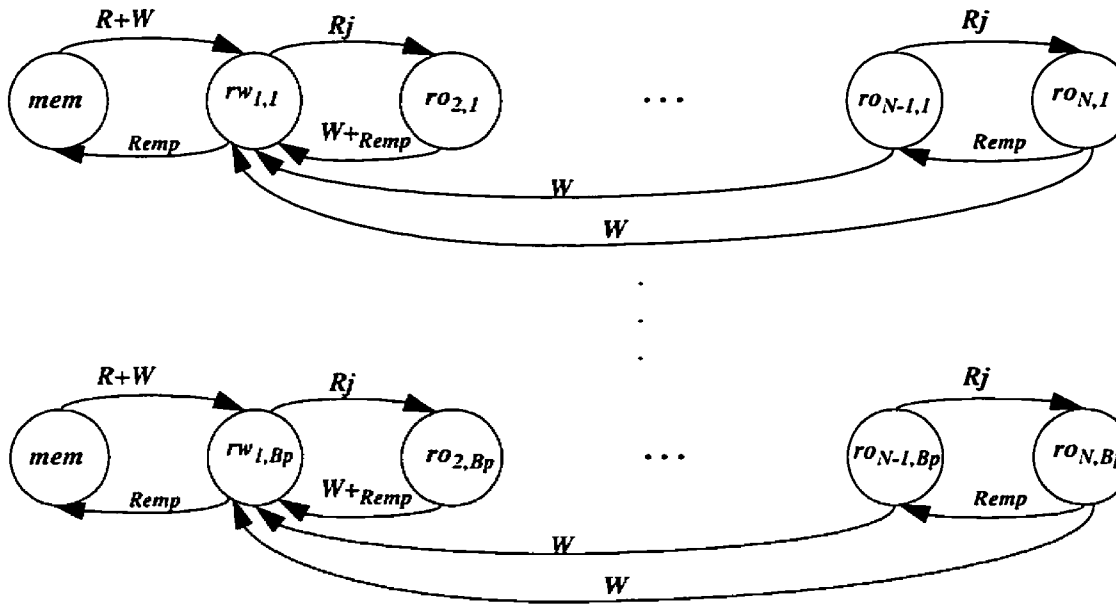
4.3.2 Modèle des données

1 - Analyse du nombre de copies

Dans le protocole d'invalidation en écriture, le nombre de copies de blocs dans le système peut être affecté par trois opérations distinctes: une opération de lecture, d'écriture ou de remplacement de bloc. Les effets de ces opérations sur le modèle sont donnés ci-dessous.

Pour une requête de lecture, si la donnée peut être satisfaite localement, le nombre de copies ne change pas; autrement, la requête doit être satisfaite par le serveur, et le nombre de copies augmente de un. Pour une requête d'écriture, il faut invalider toutes les copies disponibles sauf celle qui doit être modifiée. Ainsi, à la suite de cette opération, il ne restera plus qu'une seule copie dans le système. Dès qu'un bloc est choisi pour être remplacé par un algorithme de remplacement, le nombre de copies diminue de un. Ces variations du nombre de copies peuvent être représentées à l'aide du diagramme d'état de la figure 4.10. Ce diagramme possède les mêmes états que le modèle de mise à jour. Seulement les conditions de transition diffèrent. Sur ce diagramme, l'état *mem* indique que la donnée est disponible aux serveurs seulement. L'état *rw* signifie qu'il n'y a qu'une seule copie dans le système et que le poste qui la détient en est le propriétaire. Les autres états indiquent le nombre de copies dans le système. Il faut noter que pour N postes, le nombre de copies maximal que l'on peut avoir est N. Une transition vers la droite du diagramme s'effectue à la fin d'une opétation de lecture par un poste n'ayant pas de copie

du bloc. Les transitions vers la gauche entre les états intermédiaires s'effectuent après une opération de remplacement de bloc. De n'importe quel état *ro*, ont transite vers l'état *rw* après une opération d'écriture.



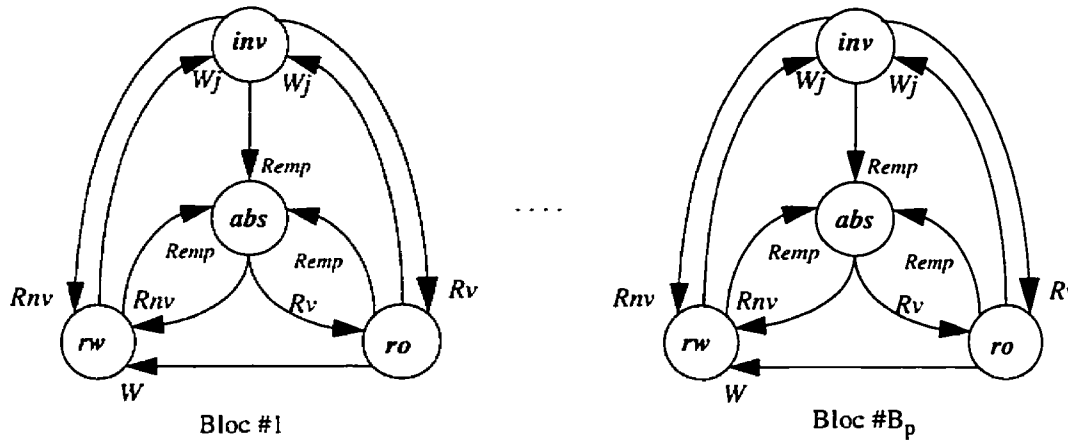
Rj: Lecture d'un bloc par un autre poste
R: Lecture d'un bloc par le processeur local
W: Écriture d'un bloc
Remp: Remplacement

Figure 4.10 Diagramme d'état pour le nombre de copies d'une donnée

2- Analyse des états des copies

Le Changement d'état est représenté par le diagramme d'état de la figure 4.11 dans lequel les états sont les états des blocs qui peuvent être *inv* (invalide), *abs* (absent), *rw* (lecture écriure) et *ro* pour (lecture seulement). Les transitions se font à la fin des requêtes de lecture et d'écriture. De l'état *abs* et *inv* on peut transiter vers l'état *rw* s'il s'agit du premier accès au bloc ou vers l'état *ro* dans le cas où il ne s'agit pas du prémier accès. Si un bloc est choisi pour être remplacé alors son état final est *abs*. Si l'état initial est *ro* alors après une opération de lecture l'état final est *rw* et si la requête est locale ou *inv* dans le cas contraire. S'il l'état inital est *rw* l'état final est *inv* après une opération d'écriture générée

par un autre poste. Si la requête est locale, l'état demeure inchangé.



W: écriture
Wj: écriture par un autre poste
R: lecture
Rv: lecture et copie valide à un autre poste
Rnv: lecture et pas de copies valides ailleurs
Remp: remplacement

Figure 4.11 Diagramme de transition d'état d'un bloc dans une mémoire locale

4.3.3 Solution du modèle

Le modèle analytique est constitué du modèle du processeur et des deux chaînes de Markov. En utilisant les mêmes hypothèses que pour le modèle de mises à jour , nous pouvons limiter nos analyses à un seul bloc.

Pour le résoudre, il faut calculer les probabilités de routage. Ainsi que les débits des chaines ouvertes. Ces quantités peuvent être obtenues à l'aide des deux chaînes de Markov. La résolution de ce modèle nous donne le débit λ_{poste} traversant le processeur.

Du diagramme d'état de la figure 4.10, nous obtenons selon l'hypothèse 13, La chaîne de Markov représentant le nombre de copies d'une seule donnée dans le système de la figure

4.12.

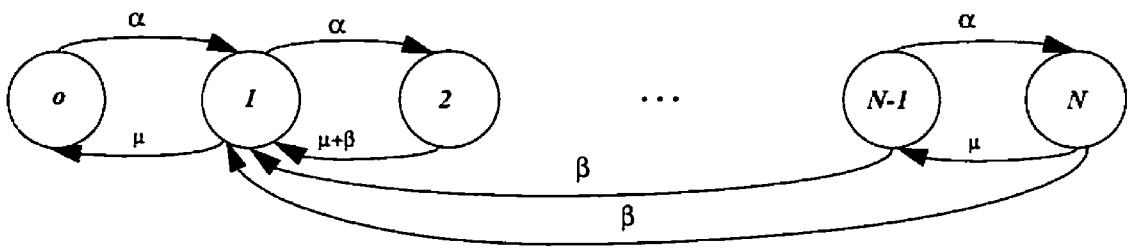


Figure 4.12 Chaîne de Markov du nombre de copies d'un bloc

Cette chaîne est à état discret et à temps continu. Comme pour le modèle de mises à jour, les taux de transition dépendent du débit λ_{poste} du modèle du processeur et de la distribution du nombre de copies.

Pour que le nombre de copies augmente, il faut faire une faute de page en lecture. Le taux α de ces transactions est donné par l'expression

$$\alpha = \lambda_{poste} \times q_j \times \pi_j[abs] \times (1 - W) \tag{4.21}$$

qui est le débit des fautes de page en lecture. Le taux β est le taux des écritures et est donné par la formule

$$\beta = \lambda_{poste} \times q_j \times W. \tag{4.22}$$

Le taux μ (figure 4.12) est le taux de remplacement des pages et est identique à celui du modèle de mises à jour. Cette chaîne est apériodique et irréductible, et peut être résolu en utilisant les équations de Chapman-Kolmogorov. De ces équations, les probabilités stationnaires $p[i]$ d'avoir i copies dans le système sont aisément extraites.

Du diagramme d'état de la figure 4.11, nous obtenons selon l'hypothèse 13, La chaîne de Markov représentant l'état d'une copie dans une mémoire locale de la figure 4.13, dont les taux de transition sont fonctions du débit des postes et des probabilités p[i].

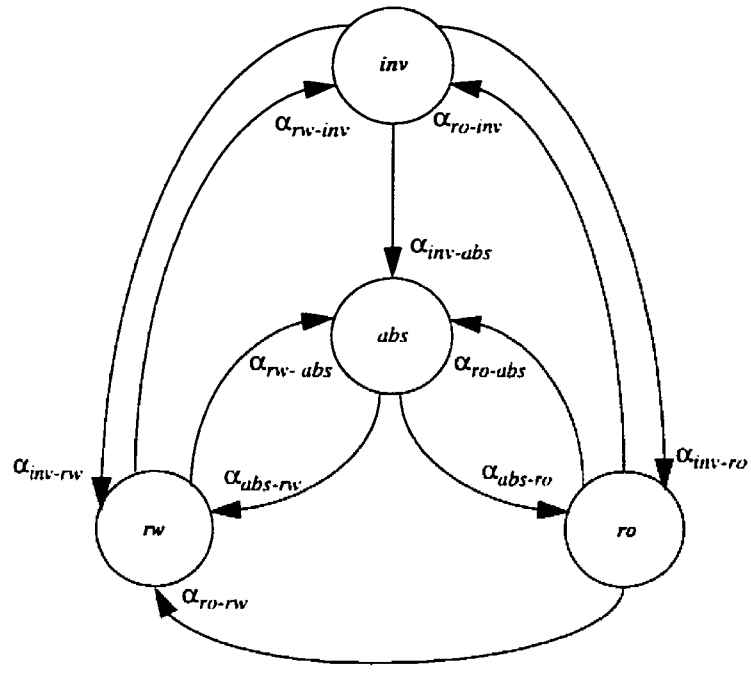


Figure 4.13 Chaîne de markov des états d'un bloc en mémoire locale

Calcul du taux de transition

Pour les transitions de *abs* à *rw*, *abs* à *ro*, *ro* à *rw*, *ro* à *abs* et *rw* à *abs* les taux de transition sont identiques à ceux du modèle de mise à jour. Pour les autres transitions, on a les quatres types de transition suivant.

transition de *inv* à *rw*

Les conditions de transition vers l'état *rw* sont les mêmes que si on partait de l'état abs. Les taux de transition sont donc identiques c'est à dire

$$\alpha_{inv-rw} = \alpha_{abs-rw} = \lambda_{poste} \times q_j \times \pi_j[abs] \times (1 - W) + \lambda_{poste} \times q_j \times W. \quad (4.23)$$

Transition de *inv* à *ro*

Ici aussi, les conditions de transition vers l'état *ro* sont les mêmes que pour les transitions

de l'état *abs* vers l'état *ro*.

Les taux de transition sont donc identiques c'est à dire

$$\alpha_{inv-ro} = \alpha_{abs-ro} = \lambda_{poste} \times q_j \times (1 - W), \quad (4.24)$$

Transition de *inv* à *abs*

Les conditions de transition sont les mêmes que si on partait des états *ro* ou *rw*. Les taux de transition sont donc identiques soit:

$$\alpha_{inv-abs} = \alpha_{rw-abs} = \alpha_{ro-abs} = \lambda_{poste} \times q_j \times (\frac{1}{C_{mem}}), \quad (4.25)$$

Transition de *rw* à *inv*

Pour transiter vers l'état *inv*, il suffit qu'un autre poste modifie la donnée. Le taux de transition est

$$\alpha_{rw-inv} = (N-1) \times \lambda_{poste} \times q_j \times \dot{W}. \quad (4.26)$$

Cette équation a déjà été rencontrée dans le modèle de mise à jour.

La formule pour la transition de l'état *ro* à *rw* est

$$\alpha_{ro-rw} = \lambda_{poste} \times q_j \times W. \quad (4.27)$$

Cette chaîne a les mêmes caractéristiques que la précédante et peut être résolue de la même manière. On obtient les probabilités stationnaires π[abs], π[rw], π[ro] et π[inv] d'avoir une copie localement dans l'un des états *abs*, *rw*, *ro* ou *inv*.

4.3.3.1 Résolution

Les probabilités de routage ont les mêmes expressions pour les deux modèles, seules leurs valeurs changent. Cependant bien que les opérations de mise à jour et d'invalidation ont des résultats différents sur les données, elles sont quantitativement assimilables par le fait qu'elles affectent les mêmes ressources et peuvent être réalisée en utilisant les mêmes techniques. Ainsi, les débits des chaînes ouvertes ont les mêmes expressions pour les deux modèles. Les deux modèles analytiques sont donc formellement identiques et peuvent être résolus en utilisant le même algorithme.

4.4 Validation des modèles

Par rapport au traitement mathématique, la simulation est plus complète en ce sens qu'elle représente fidèlement le comportement d'un système physique même dans les moindres détails sauf pour les considérations économiques. Les résultats produits par le modèle mathématique ne sont généralement valables qu'en régime permanent alors que la simulation fournit en plus des résultats en régime transitoire.

La méthode utilisée est la simulation statistique à événement discret qui utilise des modèles fondés sur les réseaux de file d'attente. Le temps simulé évolue par des pas discrets entre des instants significatifs. Ces instants correspondent à un changement d'état du système. Une tâche généralement appelée transaction évolue dynamiquement dans le système. Il existe des logiciels intégrés spécialement adaptés au traitement des modèles de systèmes informatiques. Ces logiciels sont munis de générateurs de nombres aléatoires qui

permettent de spécifier des paramètres d'entrée de la simulation. Une horloge logique permet de déclencher et d'arrêter la simulation.

4.4.1 Modèles de simulation

Afin de valider les modèles analytiques que nous avons développés au chapitre précédent, nous avons conçu un modèle de simulation qui permet d'obtenir le temps de réponse du système original. Ce modèle est représenté par un algorithme dont les organigrammes se trouvent aux figures 4.14, 4.15, 4.16 et 4.17. Nous allons le décrire dans les sections qui suivent. Nous supposons dans ce modèle que les temps de service des mémoires locales, du réseau et des SP sont constants.

4.4.1.1 Description des identificateurs du modèle de simulation

L'algorithme du modèle de simulation utilise des variables globales et des variables locales qui sont décrit ci-dessous. Il nous fournit en sortie, la puissance de calcul que nous désignons par la variable P_{sim}.

Les variables globales de l'algorithme du modèle de simulation

L'algorithme utilise les variables globales suivantes:

- le nombre de postes;
- le nombre de serveurs principaux;
- le nombre de serveurs secondaires;
- le nombre total de blocs dans le système;
- la capacité maximale d'une mémoire locale;
- le temps d'interarrivée des requêtes ;

- le temps de service moyen des mémoires locales ;

- le temps de service moyen du réseau;

- le temps de service moyen des serveurs;

- le taux d'écriture;

- l'état des blocs.

Ces variables sont résumées au tableau 4.1 qui contient le nom, le type ainsi que la description de chacune d'elles.

Tableau 4.1: Description des variables globales

Nom	Type	Description
N	Entier	Nombre total de postes de travail.
SP	Enter	Nombre total de serveurs principaux.
SS	Entier	Nombre total de serveurs secondaires.
B_p	Entier	Nombre total de bloc partagés.
C_{mem}	Entier	Capacité maximale d'une mémoire locale.
T_{arr}	Réel	Temp moyen d'interarrivée des requêtes.
$T_{mémoire}$	Réel	Temp moyen de service à la mémoire.
$T_{réseau}$	Réel	Temps moyen de service du réseau.
$T_{serveur}$	Réel	Temp moyen de service des serveurs.
ABS	Entier	État d'un bloc absent en mémoire locale.
RW	Entier	Etat d'un bloc à lecture et écriture.
RO	Entier	Etat d'un bloc à lecture seulement.
INV	Entier	Etat d'un bloc invalide.
Lecture	Entier = 0	Requête de lecture.
Écriture	Entier = 1	Requête d'écriture.
W	Réel	Taux d'écriture entrer 0.0 et 1.0.

Description des variables locales de l'algorithme du modèle de simulation

L'algorithme pour le calcul de performance du modèle de simulation à besoin des variables locales suivantes:

- le genre de requêtes qui peut être une lecture ou une écriture;
- le numéro du bloc;
- le numéro du poste;
- un compteur pour le nombre de blocs en mémoire;
- un vecteur d'état pour l'état des blocs en mémoire locale;
- le temps de réponse;
- un compteur pour le nombre de copies dans le système;
- une variable pour identifier les propriétaires des blocs;
- une variable pour identifier les serveurs principaux responsables des blocs;
- une variable pour identifier les serveurs secondaires;
- un fanion pour contrôler les remplacements;
- un fanion pour contrôler les mises à jour des copies aux serveurs principaux;
- un fanion pour contrôler les transferts de privilège;
- un fanion pour contrôler mise à jour des serveurs secondaires;
- un fanion pour contrôler les mises à jour ou les invalidations des copies aux postes.

Ces variables sont décrites au tableau 4.2 qui contient le nom, le type ainsi que la description de chacune d'elles. Pour ce qui est des types, la notation {valeur1, valeur2,..} représente un ensemble de valeurs et signifie que la variable peut prendre des valeurs de l'ensemble. La notation {ValeurFinale..ValeurFinale} représente un intervalle de valeurs et signifie que la variable peut prendre des valeurs dans cet intervalle.

Tableau 4.2: Description des variables locales de l'algotihme de simulation

Nom	Type	Descrition
Requête	Entier de valeurs { *Lecture, Écriture* }	Le genre de requête.
NumBloc	Entier de valeurs {*1..Bp*}	Numéro du bloc.
NbBloc	Tableau[*1..N*] de valeurs {*1..Cmem*}	Nombre de bloc dans chaque mémoire locale.
Etat	Tableau[*1..N*] de valeurs {*ABS, RW, RO, INV*}	Vecteur d'état des blocs dans chaque mémoire.
NbCopies	Tableau[*1..Bp*] de valeurs {*0..N*}	Nombre total de copies des blocs dans le système.
Propriétaire	Tableau[*1..Bp*] de valeurs {*1..N*}	Identité des propriétaires de chaque bloc.
ServRes	Tableau[*1..Bp*] de valeurs {*1..SP*}	Identité des serveurs principaux responsables des blocs.
Secondaire	Tableau[*1..Sp*] de valeurs {*1..SS*}	Identité des serveurs secondaires
FanRemp	Booléen	Fanion pour le déclenchement des remplacements.
FanAjServ	Booléen	Fanion pour le déclenchement de la mise à jour des serveurs principaux.
FanAjPost	Booléen	Fanion pour le déclenchement des mises à jour ou d'invalidation des copies aux poste.
FanPriv	Booléen	Fanion pour le déclenchement du transfert de privilège.
FanSec	Booléen	Fanion pour la mise à jour des serveurs secondaires.

Algorithme

Le programme est constitué de six parties ayant chacune des responsabilités bien spécifiques.

La première partie permet de calculer le temps de réponse moyen. Les ressources matérielles impliquées sont le poste de travail, le réseau, les serveurs principaux. Cette partie est représentée à la figure 4.14. Le principe est simple. Lorsqu'une requête est générée par le processeur, son temps de début est enregistré. Il en est de même lorsque la requête est satisfaite. Le temps de réponse de cette requête est calculé en enlevant le temps de début de cette requête de son temps de fin. Le temps de réponse moyen est calculé en divisant la somme des temps de réponse par le nombre de requêtes.

La seconde partie de l'algorithme simule la mise à jour ou l'invalidation des blocs dans les mémoires locales par les serveurs principaux. Il implique les mêmes ressources matérielles que le précédent sauf que l'initiateur de la requête est un serveur principal au lieu d'être un poste. Cette opération est effectuée selon la technique de mise à jour en vigeur à savoir la diffusion simulée, la diffusion véritable et le chaînage. Les détails de chacune des techniques sont décrits à la figure 4.15. Dans le cas du chaînage, les postes impliqués sont visités un par un. Dans le cas d'une diffusion véritable, un seul client est envoyé au réseau. À la sortie du réseau, on génère de clients supplémentaires de manière à avoir autant de clients qu'il y a de postes impliqués. Chaque client est par la suite envoyé à un poste. Lorsque l'opération est terminée au niveau du poste, on détruit tous les clients supplémentaires. Dans le cas d'une diffusion simulée, l'opération de création se fait au niveau du serveur c'est à dire avant le transfert au réseau.

La troisième partie sert à simuler la mise à jour des serveurs secondaires par les serveurs principaux. Pour une donnée, cette partie implique le serveur principal responsable du bloc, le réseau ainsi que tous les serveurs secondaires. Elle est représentée à la figure 4.16 . Un client initialement au serveur principal doit consulter selon la technique utilisée, tous les serveurs secondaires.

La quatrième partie sert à simuler le remplacement des blocs au niveau des mémoires locales. Le client qui réalise cette opération est toujours actif et attend que le fanion soit actif. Par la suite, il se rend au serveur principal mettre à jour le nombre de copies. Lorsque l'opération est terminée, le fanion est désactivé et le client attend une prochaine activation. Cette partie est représentée à la figure 4.17a.

La cinquième partie sert à simuler la mise à jour des blocs aux serveurs principaux par les postes de travail. Comme dans le cas précédent, le client qui réalise cette opération est toujours actif et attend que le fanion soit actif. Par la suite, il se rend au serveur principal. Lorsque l'opération est terminée, le fanion est désactivé et le client attend une prochaine activation. Cette partie est représenté à la figure 4.17b.

La sixième et dernière partie de l'algorithme sert à simuler le transfert de privilège. Elle implique pour un bloc donné le serveur principal responsable, le réseau et le poste propriétaire du bloc. Elle est représentée à la figure 4.17c. Son fonctionnement obéit aux même principe que le précédant, sauf que le client est initialement au serveur principal est doit se rendre au poste propriétaire du bloc.

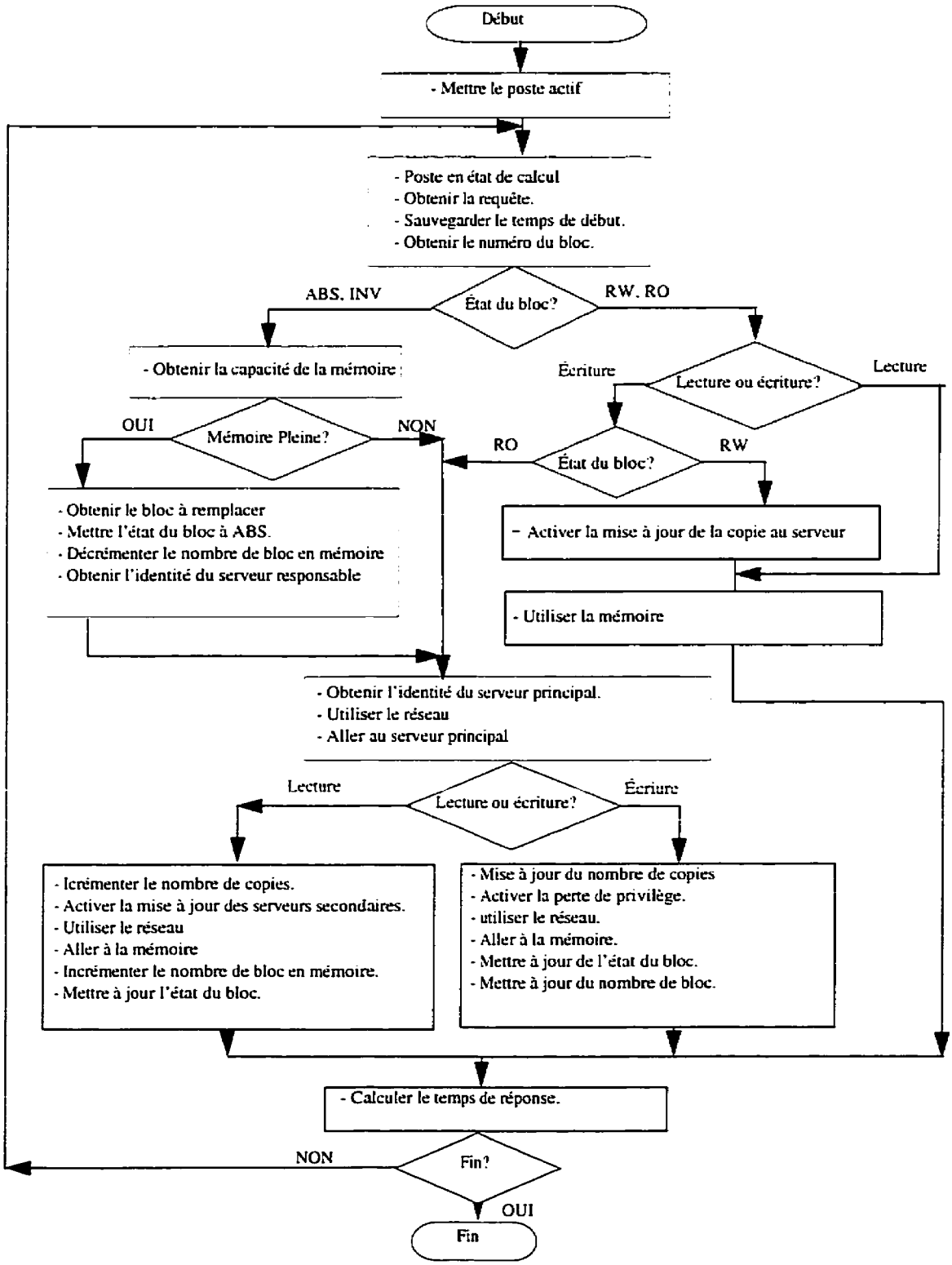


Figure 4.14 Modèle pour le calcul du temps de réponse

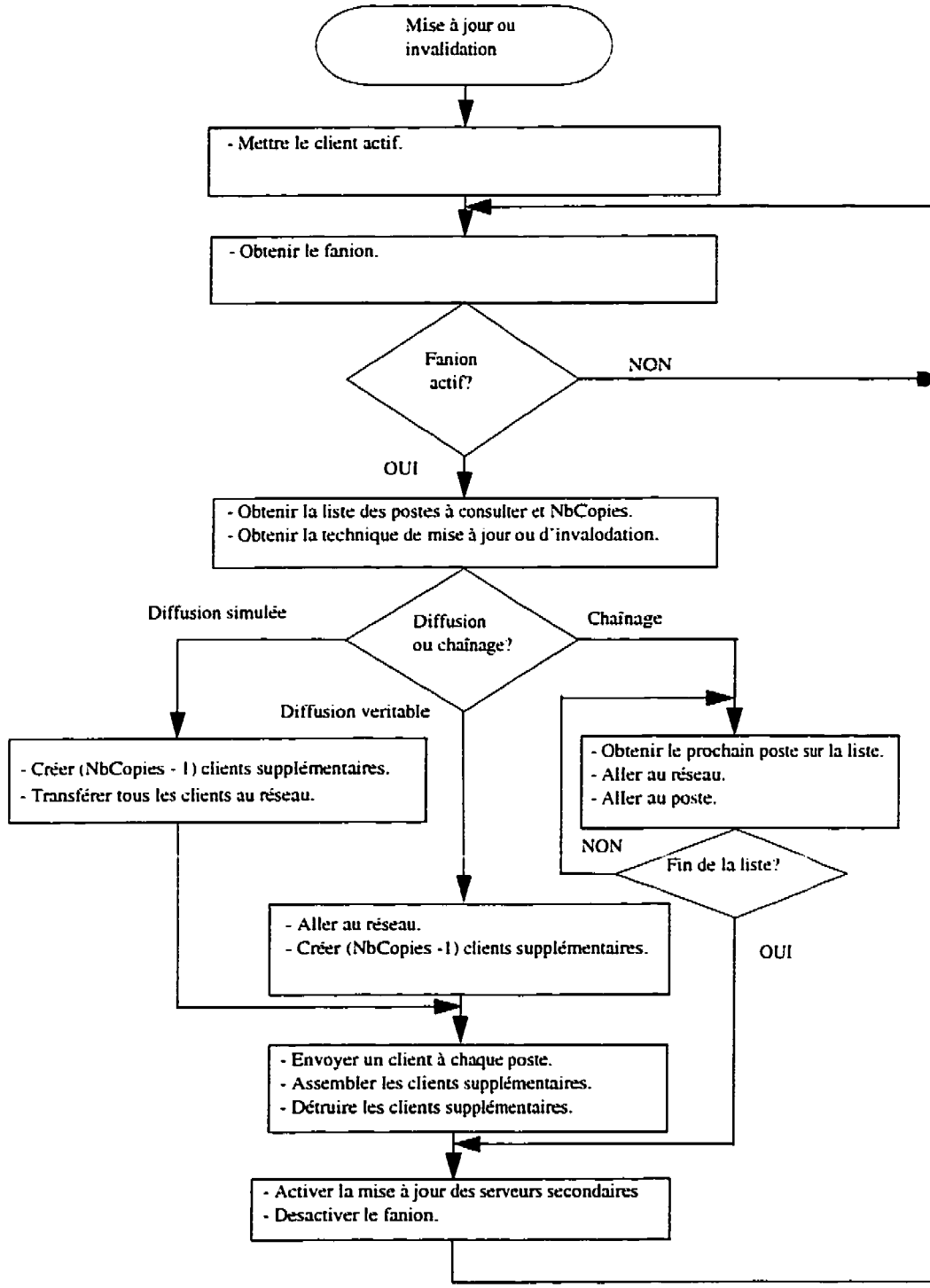


Figure 4.15 Modèle pour la mise à jour ou l'invalidation des postes

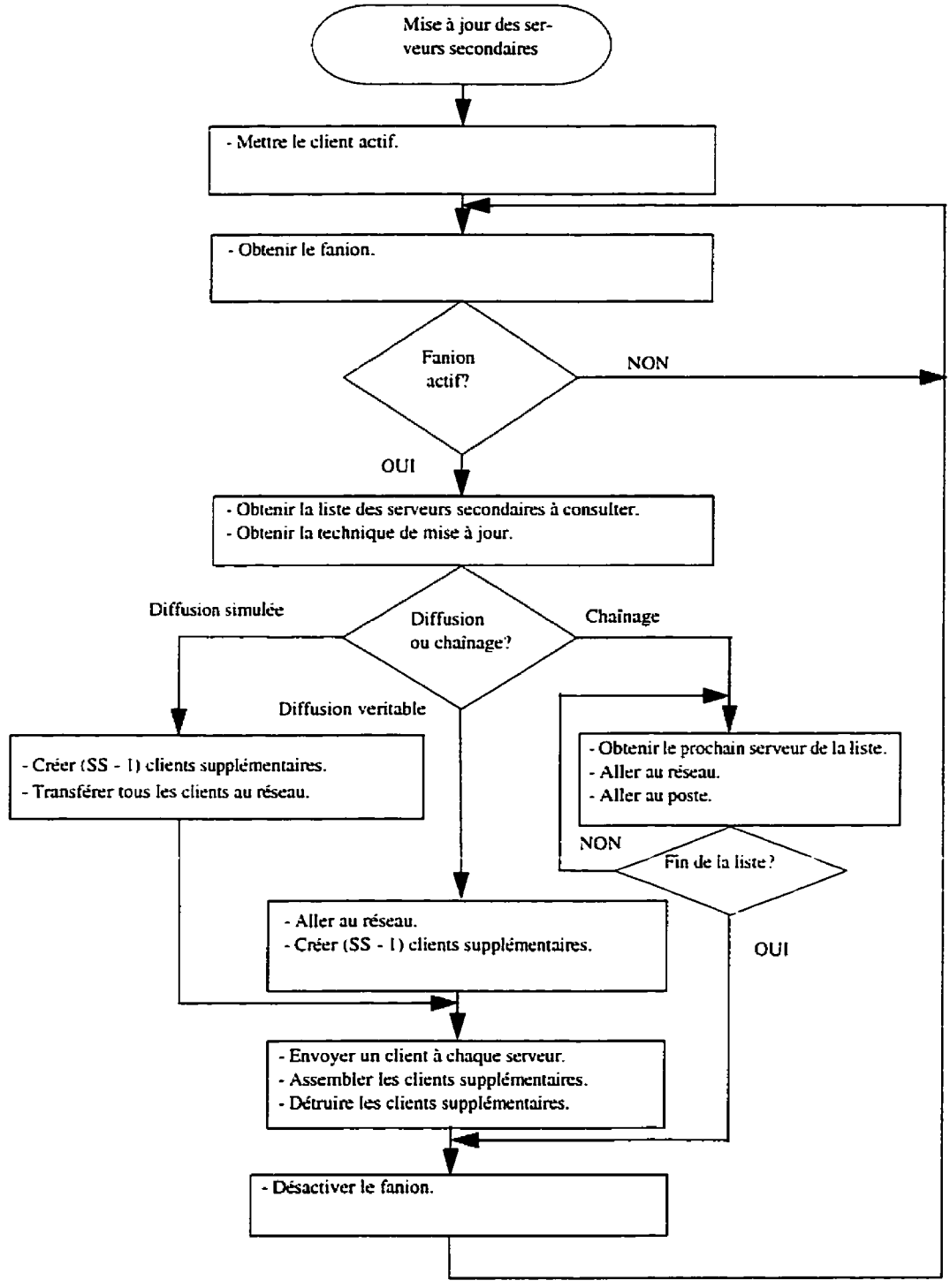


Figure 4.16 Modèle pour la mise à jour des SS

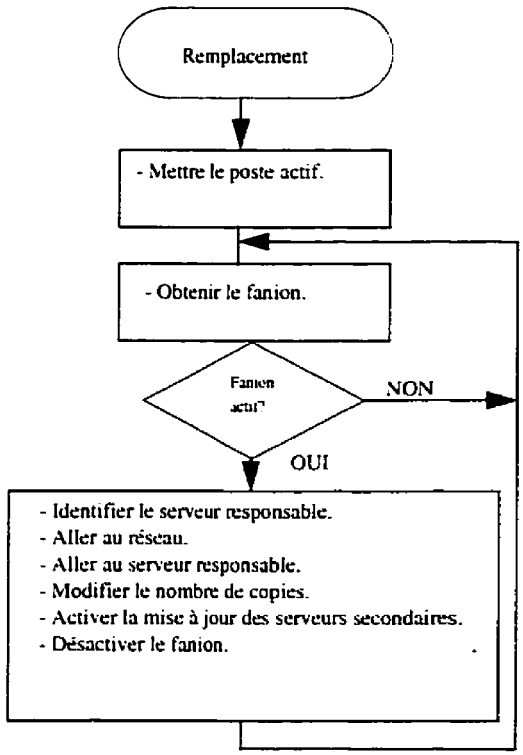


Figure 4.17a Remplacement

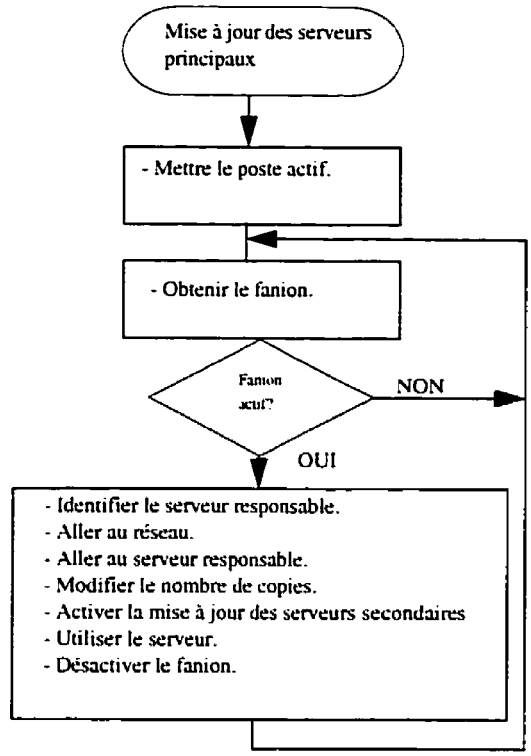


Figure 4.17b Mise à jour des SP

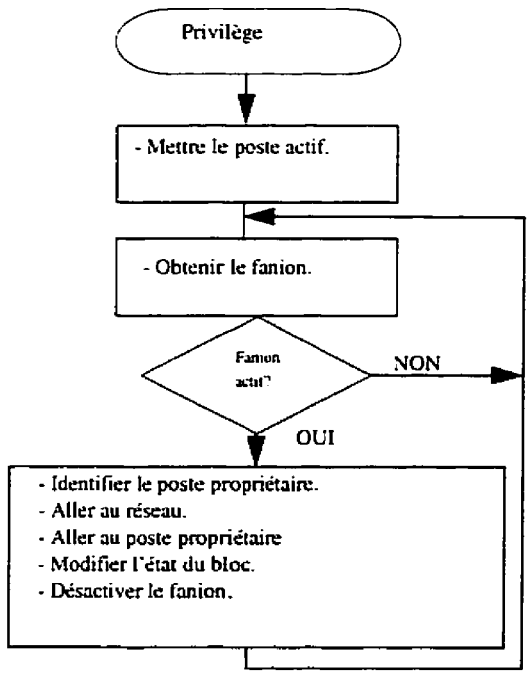


Figure 4.17c Transfert de privilège

CHAPITRE 5

RÉSULTATS ET DISCUSSIONS

Au chapitre précédent, nous avons développé les modèles analytiques ainsi que les modèles de simulation de notre système. La résolution de ces modèles nous a donné la puissance de calcul qui tient compte non seulement du nombre de postes, mais également de l'effet des protocoles de cohérence. Dans ce chapitre, notre but est d'abord de valider les modèles, puis d'étudier l'effet d'un certain nombre de paramètres architecturaux sur les performances du système. Il s'agit des paramètres suivants:

- le temps d'interarrivée;
- la capacité des mémoires locales;
- le nombre de serveurs principaux;
- le nombre de serveurs secondaires;
- la vitesse du réseau.

Pour réaliser cette étude, nous utiliserons le modèle analytique. Le modèle de simulation servira à valider le modèle analytique.

Dans la suite de ce mémoire, nous utiliserons le terme unité de temps (UT) pour désigner l'unité de grandeur des temps de service car ces paramètres ont des valeurs inconnues et dépendent non seulement de l'architecture du système mais également de la nature de l'application.

5.1 Validation des modèles

5.1.1 Choix des paramètres

Pour valider les modèles, nous avons utilisé pour des raisons de simplicité, des modèles de

simulation réduits. En effet, chaque site a besoin d'une structure de données en tableau pour représenter le répertoire d'états ou de copies mais également d'un mécanisme d'indexation pour accéder aux différents éléments du tableau. Or GPSS n'offre pas cette possibilité [Karian 90, Schriber 90]. En l'absence de cette structure, il est difficile de construire un modèle opérant sur une grande quantité de données et comportant un nombre important de postes. Nous avons retenu les deux configurations de la table 5.1.

Tableau 5.1 Paramètres de simulation

Paramètres	Système faiblement chargé(système #1)	Système fortement chargé (système #2)
T_{arr} *en UT*	5	2
$T_{mémoire}$ *en UT*	0.1	0.1
$T_{réseau}$ *en UT*	0.1	0.1
$T_{serveur}$ *en UT*	0.1	0.1
N	3	3
SP	2	2
SS	1	1
B_p	2	2
C_{mem}	1	1

Il s'agit de deux systèmes ayant chacun trois postes de travail, trois serveurs dont un serveur secondaire et deux serveurs principaux. Les systèmes sont complets en ce sens qu'ils permettent de faire la différence entre toutes les techniques de mise à jour ou d'invalidation étudiées à savoir, la diffusion simulée, la diffusion véritable ainsi que le chaînage. Les temps de service des mémoires locales, du réseau ainsi que des serveurs sont tous de 0.1 UT. La capacité de la mémoire locale est de un bloc et le nombre de blocs partagés est deux. Ce choix nous assure que le débit de remplacement des blocs ne sera pas nul car le nombre de blocs partagés est supérieur à la capacité des mémoires locales.

La première configuration a un temps d'interarrivée de 5.0 UT et représente un système faiblement chargé (car T_{arr} élevé). Cette configuration sera appelée système #1. La seconde a un temps d'interarrivée de 2.0 UT et représente un système fortement chargé (car T_{arr} faible). Cette configuration sera appelée système #2.

Nous avons calculé pour plusieurs taux d'écriture (W), la différence (que nous notons *Écart*) entre la puissance de calcul (P_{cal}) fournie par chaque modèle de simulation (P_{sim}) et celle fournie par son modèle équivalent analytique (P_{ana}). Les mesures de performance sont présentées à la section qui suit.

5.1.2 Résultats numériques

Les résultats numériques obtenus sont reportés dans les tableaux de l'annexe 1 et les graphiques associés sont représentés aux figures 5.1 à 5.6 qui suivent.

On constate que lorsque les techniques utilisées sont la diffusion simulée et la diffusion véritable, les écarts entre le modèle de simulation et son équivalent analytique sont faibles et les courbes correspondantes sont presque confondues. Cette remarque est aussi valable dans le cas du chaînage, mais seulement lorsque le système est faiblement chargé. Par contre lorsque la technique du chaînage est utilisée dans le système #2, les écarts sont plus importants. On note dans ce cas un écart maximal de 107.7×10^{-3} (voir tableau VI de l'annexe 1) comparé à 3.2×10^{-3} (voir tableau III de l'annexe 1) dans le cas précédant. Cependant, ces écarts demeurent négligeables. On en conclut que les modèles analytiques obtenus sont une représentation de notre système réel.

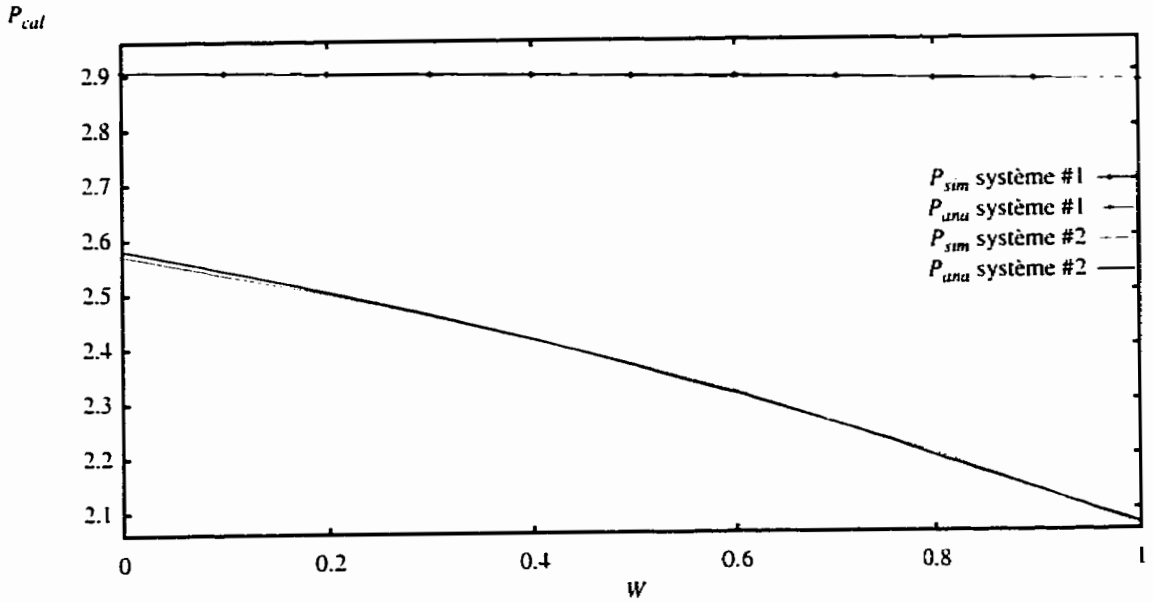


Figure 5.1 Validation des modèles pour une mise à jour simulée

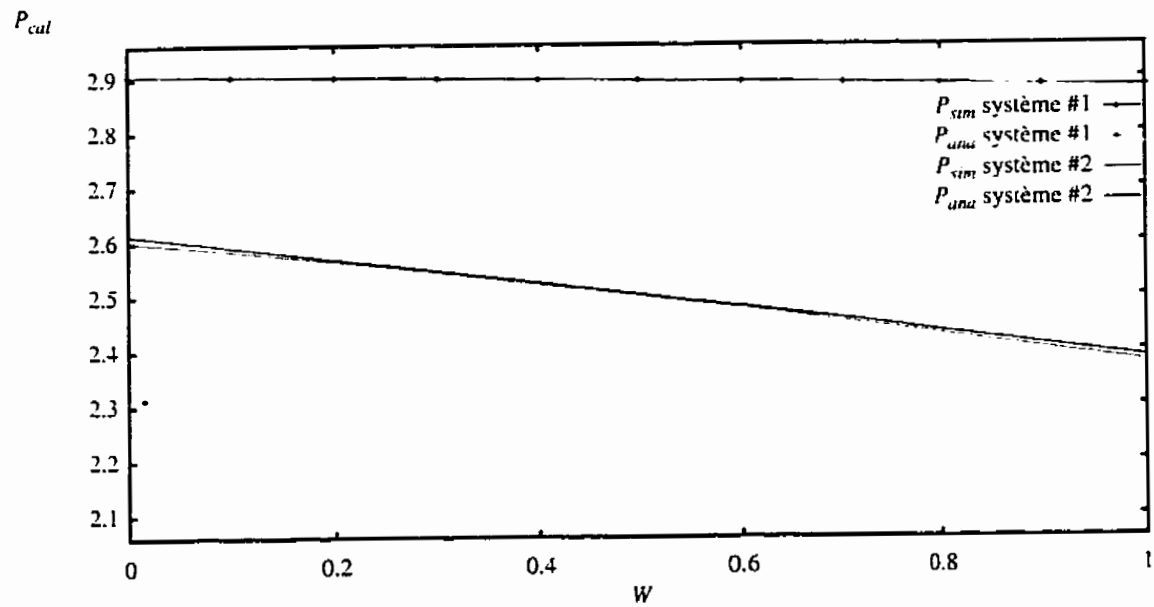


Figure 5.2 Validation des modèles pour une mise à jour véritable

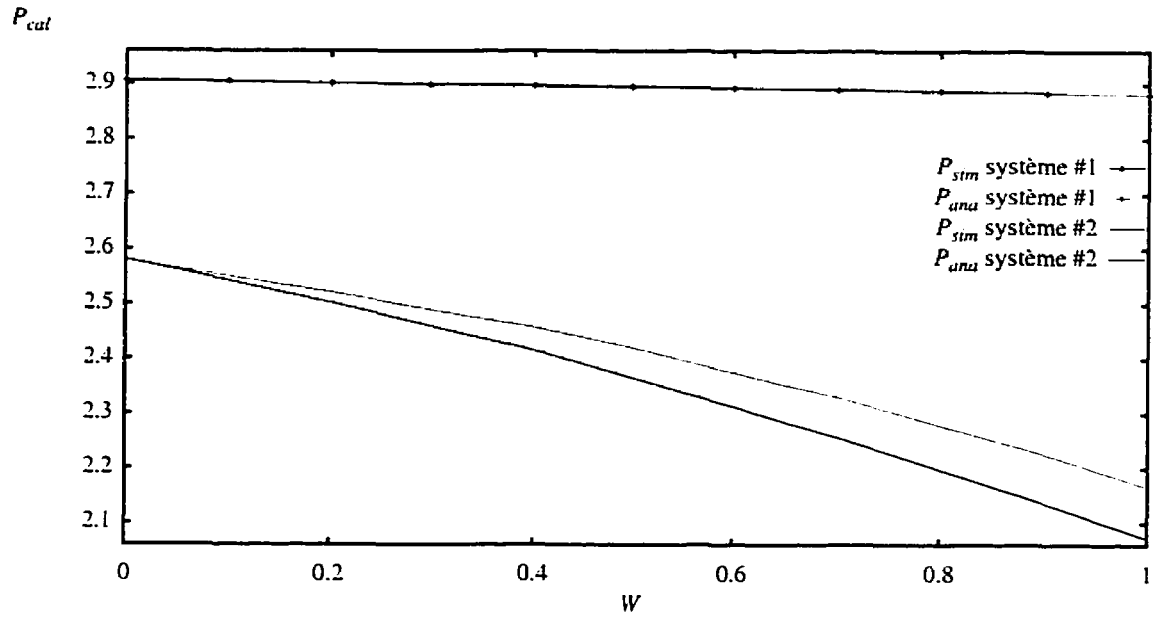


Figure 5.3 Validation des modèles pour une mise à jour par chaînage

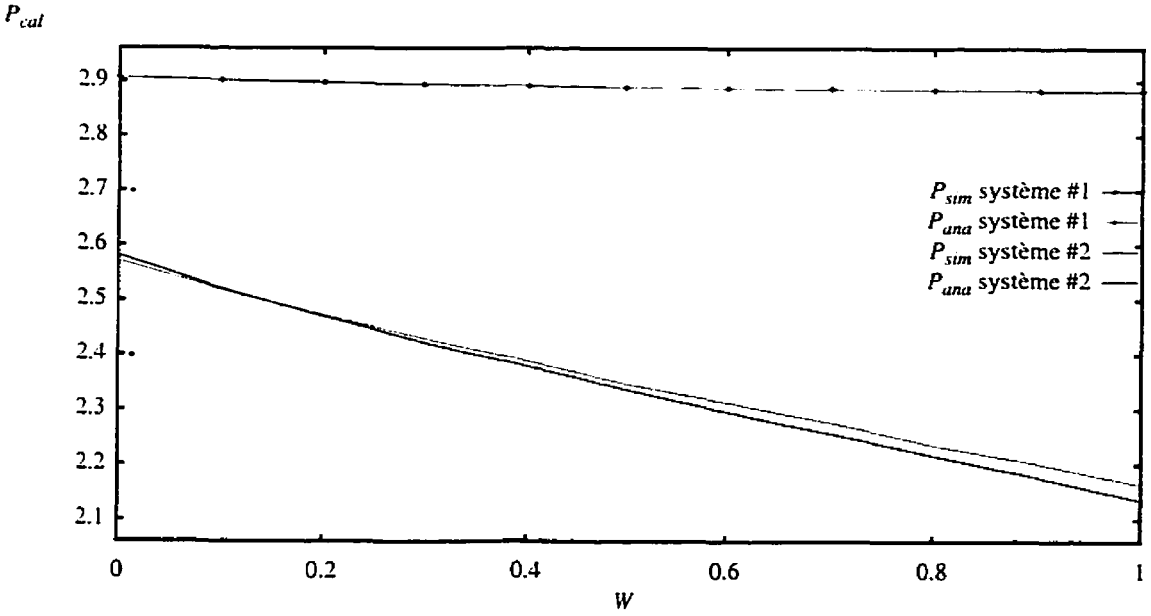


Figure 5.4 Validation des modèles pour une invalidation simulée

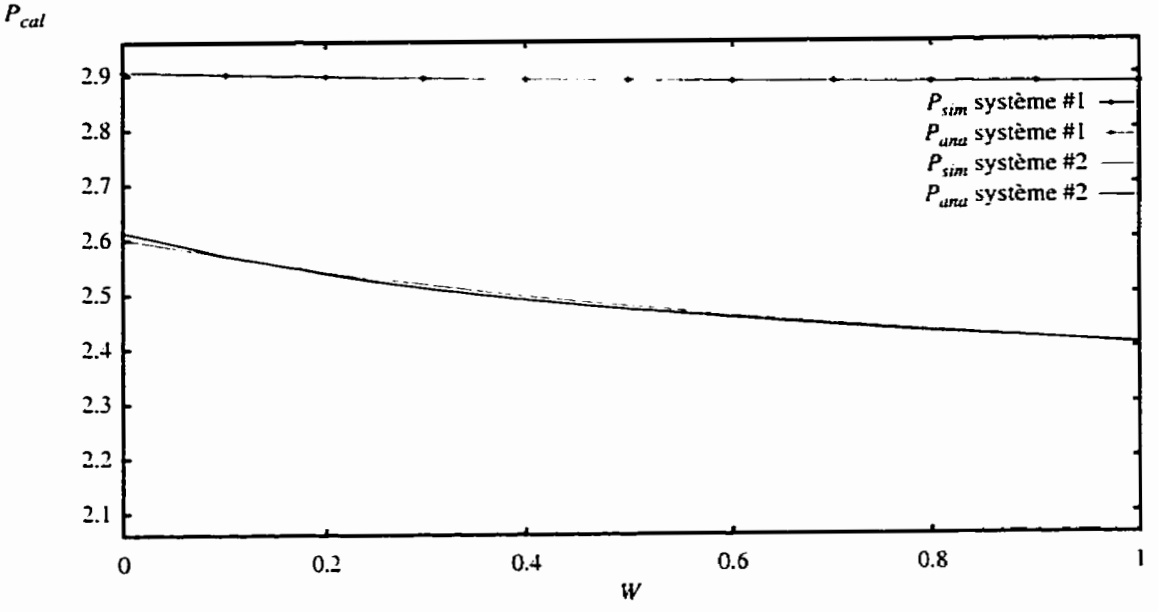


Figure 5.5 Validation des modèles pour une invalidation véritable

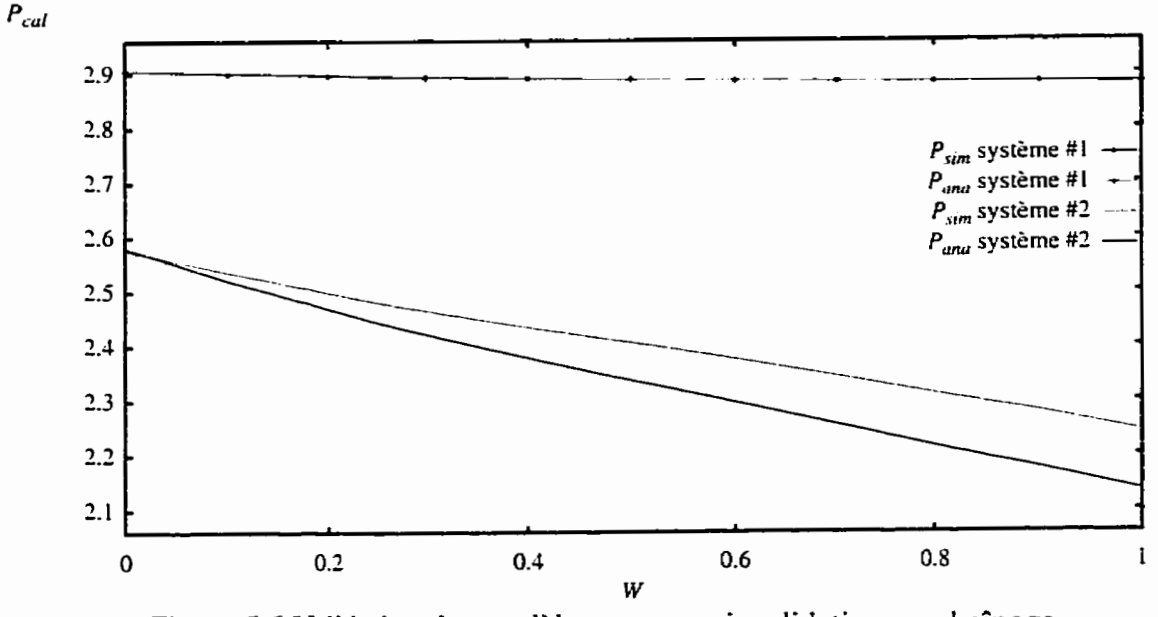


Figure 5.6 Validation des modèles pour une invalidation par chaînage

5.2 Étude du système sous certaines conditions de charges

Dans les sections suivantes, nous allons étudier l'effet de la variation de certains paramètres sur la puissance de calcul.

5.2.1 Choix du système de référence

Nous avons retenu comme système de référence, un système dont les paramètres sont ceux indiqués au tableau 5.2.

Tableau 5.2: Paramètres du système de référence

Paramètres	Valeurs
N	30
SP	5
SS	1
B_p	32
C_{mem}	8
$T_{mémoire}$	0.01
$T_{serveur}$	0.01
$T_{réseau}$	0.0015
T_{arr}	1.0

Il s'agit d'un système de taille moyenne car le nombre de sites n'est pas élevé ($N = 30$, $SP = 2$ *et* $SS = 5$). Le réseau est à haut débit car son temps de service est faible comparé aux temps de service des mémoires locales et des serveurs. Le système est fortement chargé car le temps d'interarrivée est faible. Chaque poste de travail exécute localement un

algorithme de remplacement de blocs car $B_p > C_{mem}$.

5.2.2 Performances du système de base

Nous avons calculé la puissance de calcul du système précédant pour trois valeurs du taux d'écriture: 0.3, 0.5 et 0.9. Les résultats sont représentés aux figures 5.7, 5.8 et 5.9 ci-dessous.

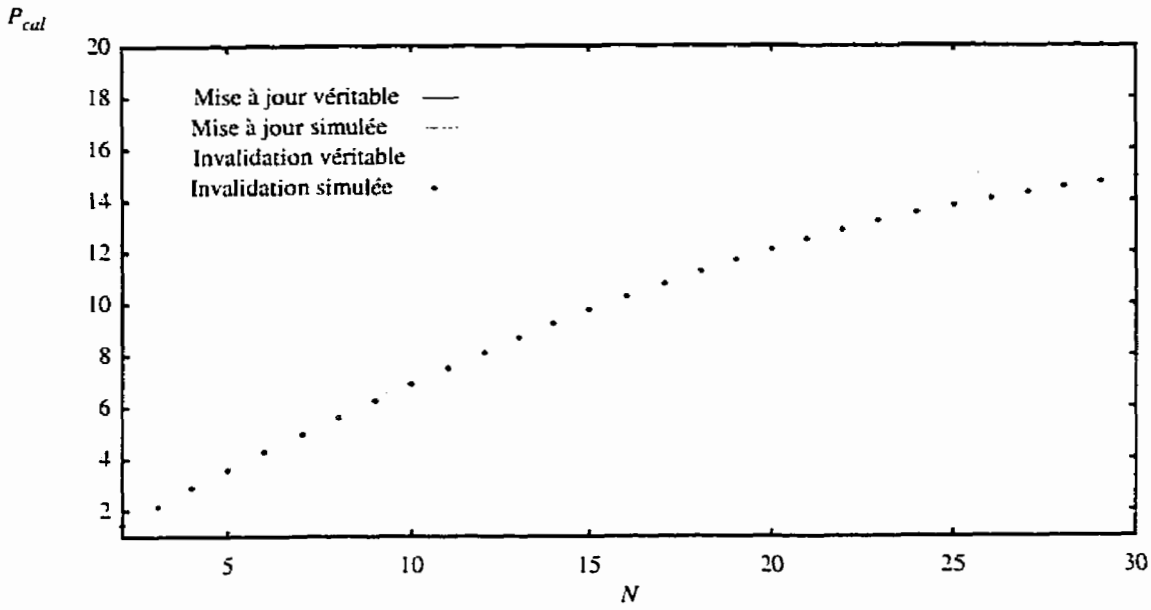


Figure 5.7 Performance du système pour un taux d'écriture faible ($W = 0.3$)

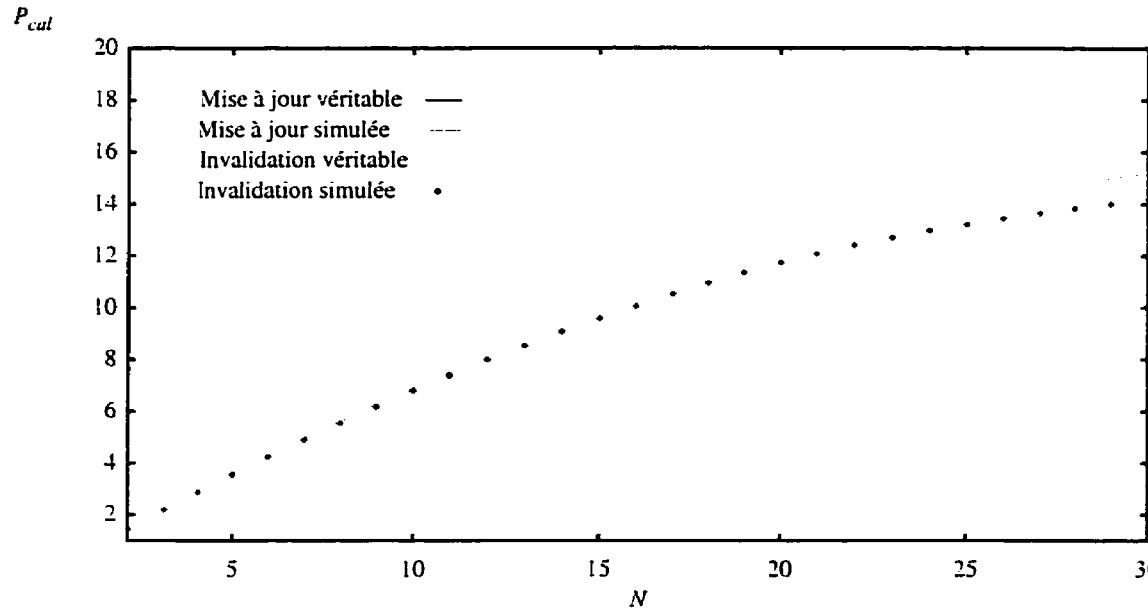


Figure 5.8 Performance du système pour un taux d'écriture moyen ($W = 0.5$)

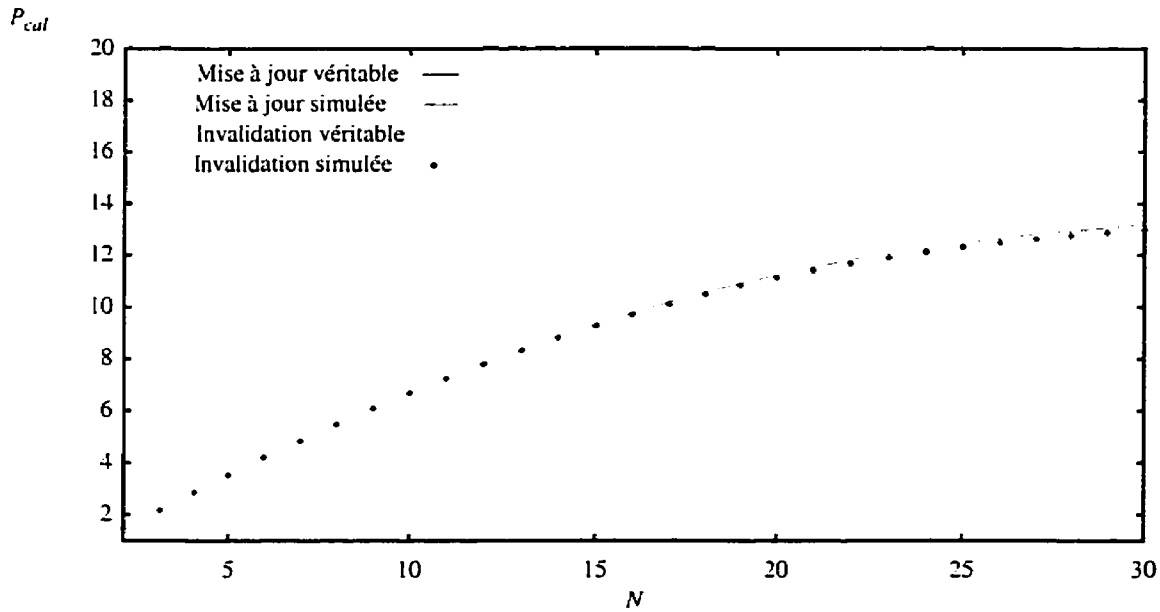


Figure 5.9 Performance du système pour un taux d'écriture élevé ($W = 0.9$)

On constate que les performances du système sont indépendantes de la technique de diffusion utilisée; c'est à dire que les résultats obtenus dans le cas d'une diffusion véritable sont identiques à ceux obtenus dans le cas d'une diffusion simulée. On constate aussi que lorsque le taux d'écriture est faible, le protocole de mise à jour en écriture est plus performant que le protocole d'invalidation. Cependant, au fur et à mesure que le taux d'écriture augmentent, la puissance de calcul diminue, la différence entre les deux protocoles diminue également. Cet écart devient nulle lorsque le taux d'écriture est égal à 0.9. D'autre part, la puissance de calcul varie d'abord linéairement en fonction du nombre de poste, puis sature à partir d'un certain seuil. La valeur de ce seuil est plus élevé dans le cas du protocole de mise à jour et diminue lorsque le taux d'écriture augmente.

5.2.3 Effet du temps d'interarrivée

Pour l'étude de ce paramètre, nous avons changé le temps d'interarrivée des requêtes de la configuration de référence de 1.0 à 20.0 UT.

Les courbes de la puissance de calcul en fonction du nombre de postes se trouvent à la figure 5.10.

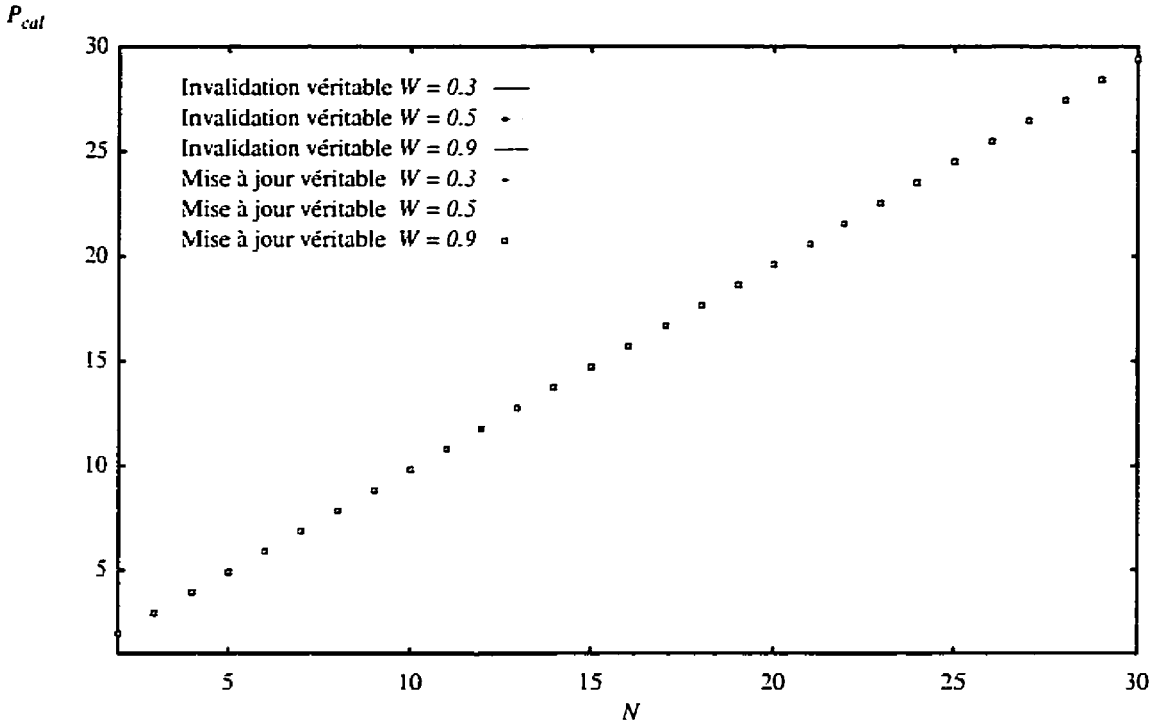


Figure 5.10 Effet du temps du temps d'interarrivée *Tarr = 20*

On constate que la puissance de calcul augmente linéairement en fonction du nombre de poste. D'autre part. les performances sont indépendantes du protocole et du taux d'écriture. Ceci est dû au fait que les postes sollicitent moins les serveurs.

Recommandations

Il est plus intéressant de concevoir un système ayant un temps de calcul très élevé, car ses performances sont indépendantes du protocole utilisé. Mais hélas, ce paramètre ne dépend pas de l'architecture du système mais plutôt de la nature de l'application. Cependant, si la nature de l'application est connue, et qu'il est possible d'évaluer l'ordre de grandeur du temps de calcul, il faudrait utiliser le protocole de mise à jour en écriture lorsque cette quantité est faible, et dans le cas contraire, n'importe lequel des deux protocoles.

5.2.4 Effet de la taille de la mémoire locale

Dans ce cas, le système à analyser est identique au système de référence, sauf que la capacité de chaque mémoire locale C_{mem} passe de 8 à 30 blocs.

Les courbes de la puissance de calcul en fonction du nombre de postes sont représentées aux figures 5.11 et 5.12. On constate que, dans le cas du protocole d'invalidation en écriture (figure 5.12), le gain de performance est négligeable quel que soit le taux d'écriture. Ceci est dû au fait que, l'augmentation de la capacité des mémoires a pour conséquence, l'augmentation du nombre de blocs en mémoire. Cependant, dès que le taux d'écriture est non nul, ces blocs transitent à l'état INV après une requête d'écriture. Pour obtenir un bloc, il faut toujours aller au serveur puisqu'ils ne sont pas disponibles localement. Ce qui augmente le temps de réponse et par conséquent, diminue la puissance de calcul. Par contre, dans le cas du protocole de mise à jour en écriture (figure 5.11), on note un gain considérable de performance. Ce gain est d'autant plus élevé que le taux d'écriture est faible. Ceci s'explique par le fait que, lorsque les capacités des mémoires locales augmentent, le nombre de blocs disponibles localement devient plus grand, ce qui diminue le nombre de fautes de bloc. Or lorsque le taux d'écriture est élevé, le nombre de copies à mettre à jour est plus important, d'où l'augmentation du temps de réponse.

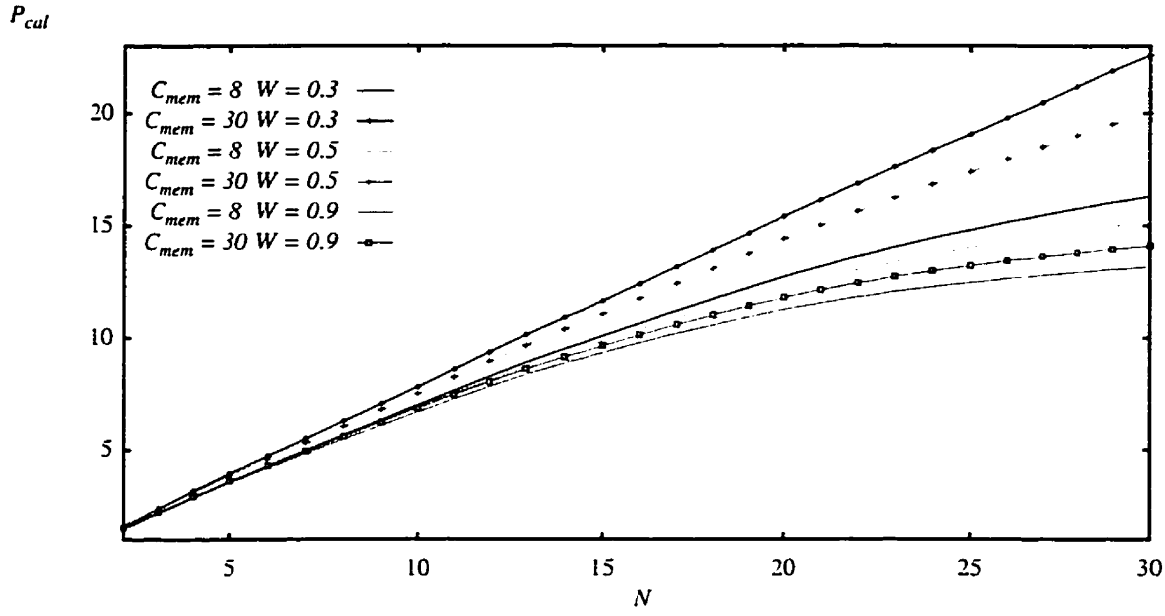


Figure 5.11 Effet des mémoires locales pour une mise à jour véritable

Recommandations

Dans les applications où le nombre de blocs est très élevé comparé à la taille des mémoires locales, le protocole d'invalidation est préférable. Dans le cas contraire, chacun des deux protocoles peut être utilisé.

5.2.5 Effet du nombre des serveurs principaux

Nous comparons notre système de référence avec deux systèmes ayant respectivement 30 et 50 serveurs principaux.

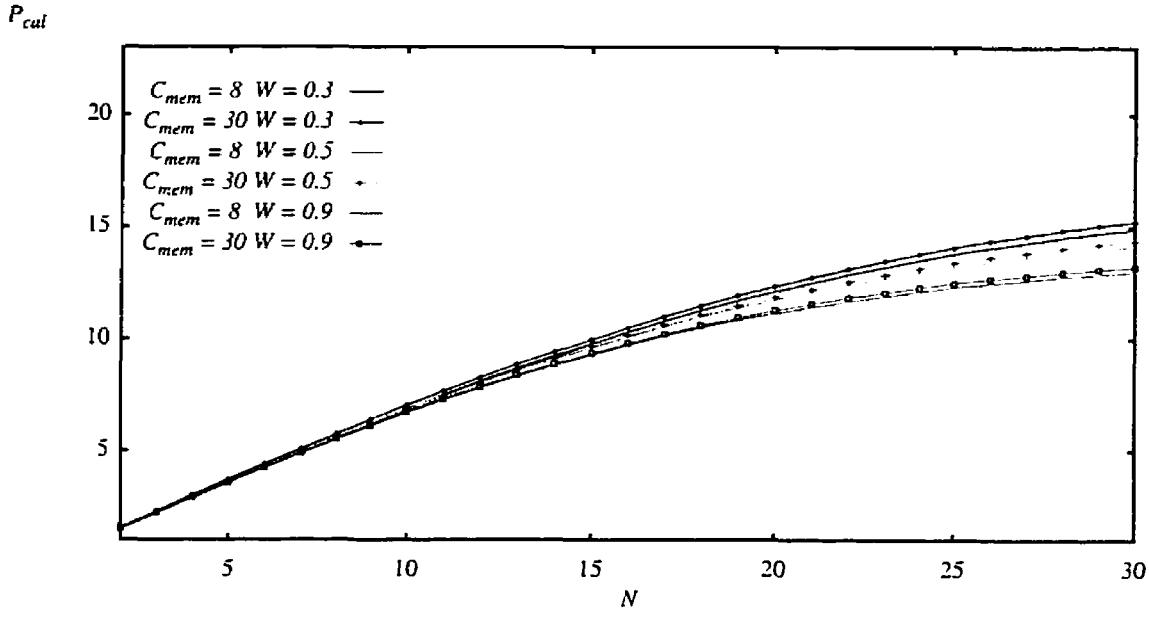


Figure 5.12 Effet des mémoires locales pour une invalidation véritable

Les courbes obtenues sont représentées aux figures 5.13 à 5.16. On constate que la puissance de calcul varie linéairement en fonction du nombre de poste; les performances s'améliorent et sont indépendantes des protocoles et des taux d'écritures. Ceci est dû au fait que lorsque le nombre de serveurs principaux est petit, ceux-ci constituent un goulot d'étranglement. Par contre leur augmentation vient éliminer ce goulot d'étranglement. Cependant si on continue à augmenter le nombre de serveurs principaux, les performances du système demeurent inchangées. Ceci est dû au fait qu'il n'existe plus de goulot d'étranglement.

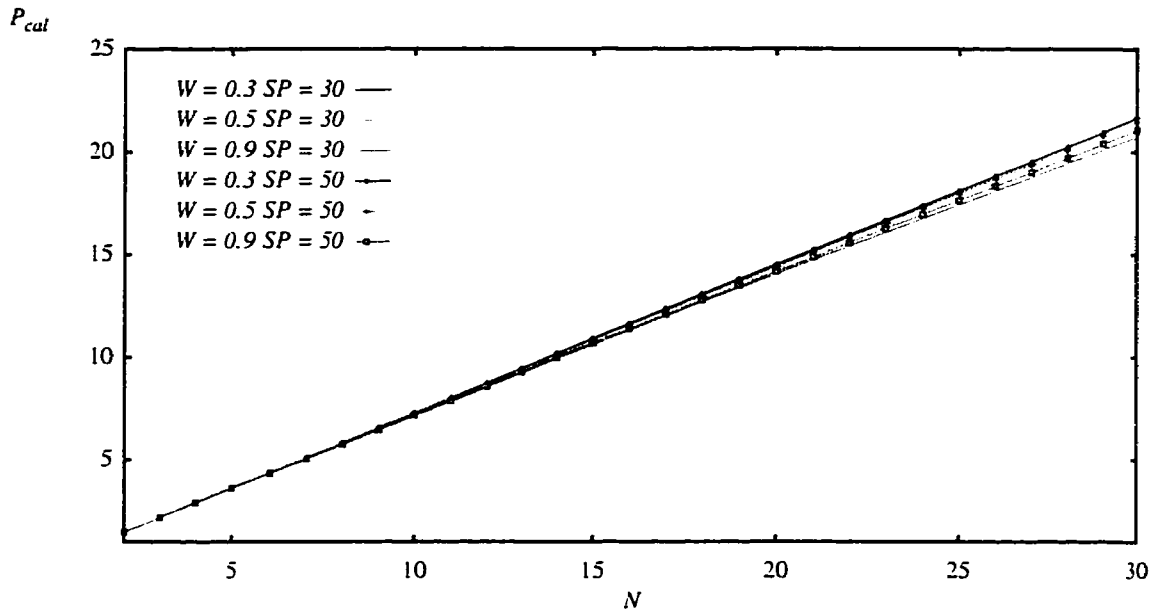


Figure 5.13 Effet du nombre de serveurs principaux pour une mise à jour véritable

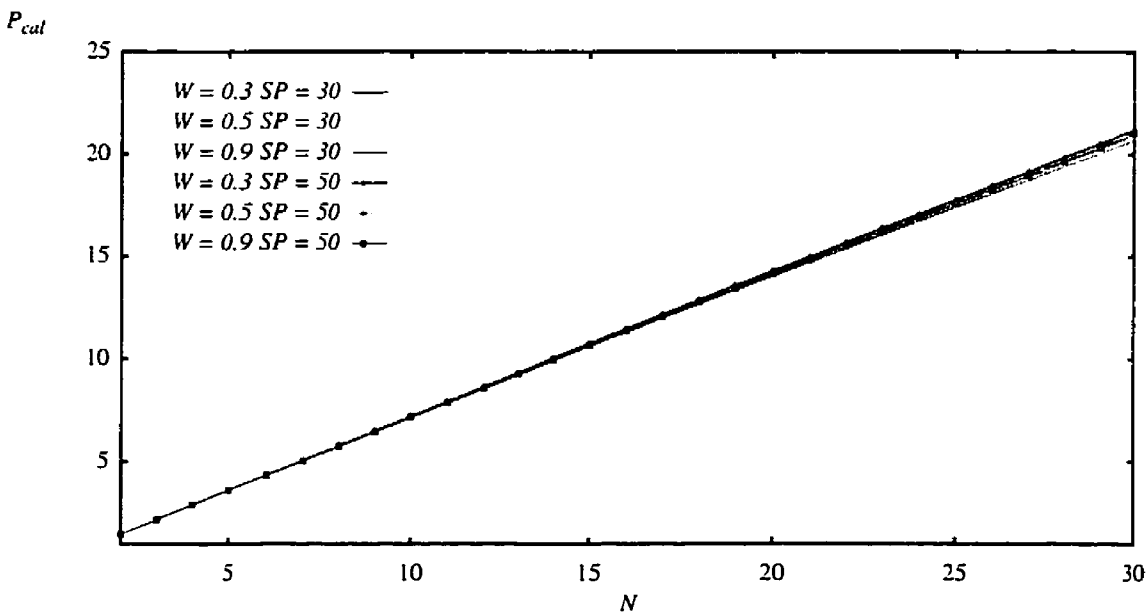


Figure 5.14 Effet du nombre de serveurs principaux pour une invalidation véritable

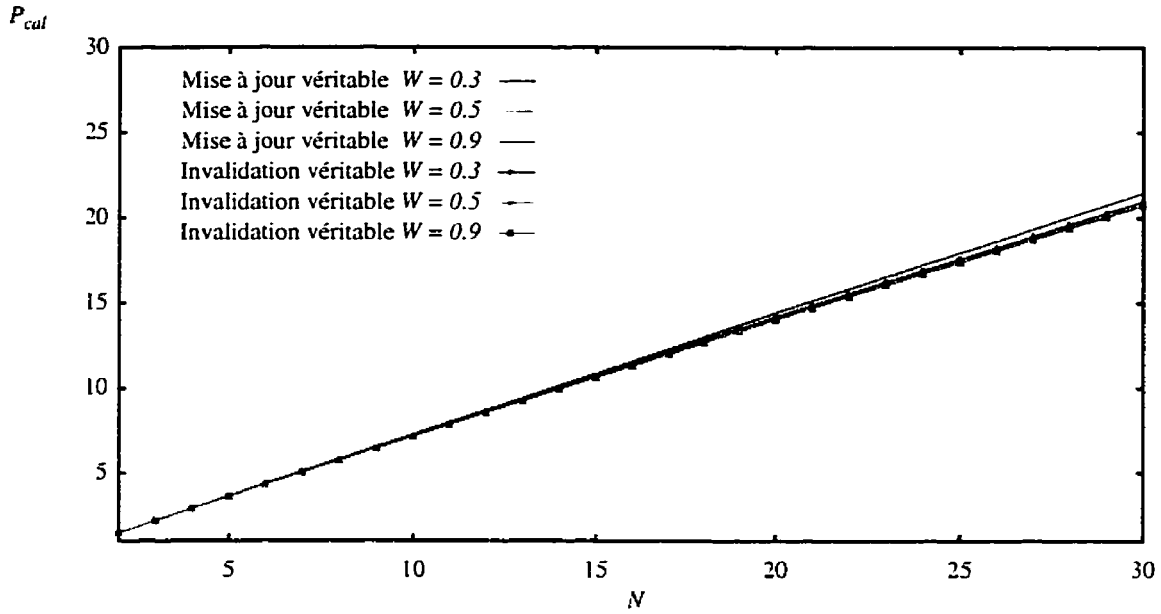


Figure 5.15 Comparaison des protocoles pour $SP = 30$

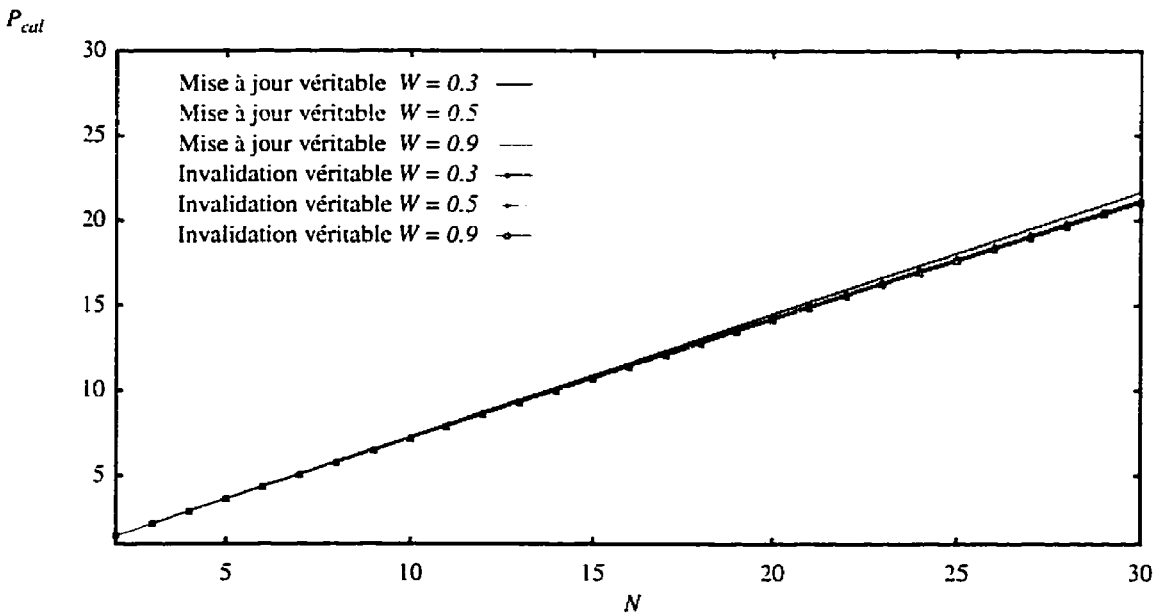


Figure 5.16 Comparaison des protocoles pour $SP = 50$

Recommandations

Lorsqu'on est limité par le nombre de serveurs principaux, il est préférable d'utiliser le protocole de mise à jour en écriture. Dans le cas contraire, n'importe lequel des deux protocoles peut être utilisé. Pour réaliser le choix du nombre de serveurs principaux, il faut partir d'une configuration ayant 1 serveur. Puis il faut augmenter progressivement leur nombre jusqu'à ce que les performances cessent de s'améliorer.

5.2.6 Effet du nombre des serveurs secondaires

Pour voir l'effet du nombre de serveurs secondaires sur les performances globales de notre système réparti, nous augmentons le nombre de serveurs secondaires de notre système de référence de 1 à 2.

Les courbes obtenues sont représentées aux figures 5.17 à 5.19. On peut constater que lorsque le nombre de serveurs secondaire augmente, deux phénomène se produisent. Premièrement, la puissance de calcul diminue. Ceci s'explique par le trafic généré entre les serveurs principaux qui vient augmenter le temps d'attente au niveau des serveurs et du réseau. Par conséquent, la puissance de calcul diminue. Deuxièmement, le nombre de postes diminue. Ceci est dû au fait que, l'utilisation du serveur par l'ensemble des chaînes ouvertes devient supérieur à 1. Théoriquement, la puissance de calcul tombe à 0. Cette limitation du nombre de postes est d'autant plus sévère que le taux d'écriture augmente et est identique pour les deux protocoles.

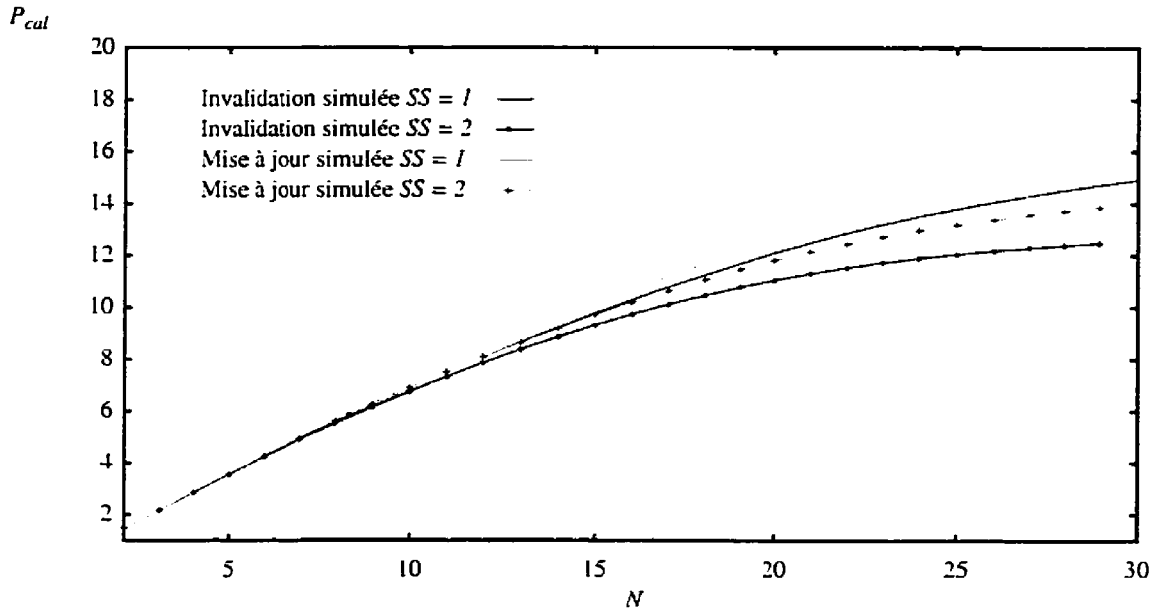


Figure 5.17 Effet du nombre de serveurs secondaires: $W = 0.3$

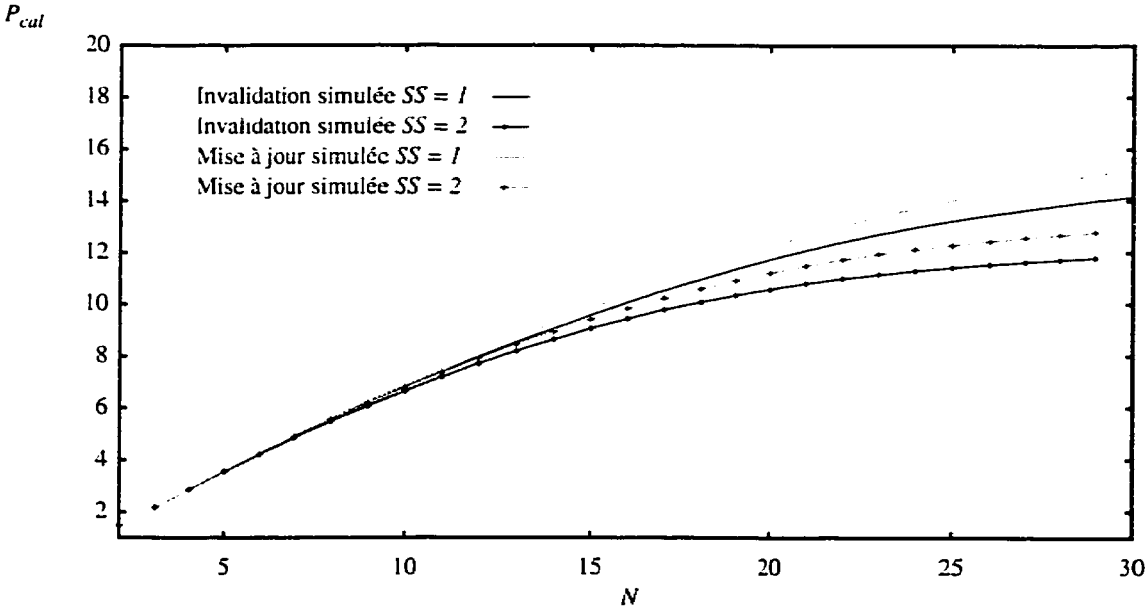


Figure 5.18 Effet du nombre de serveurs secondaires: $W = 0.5$

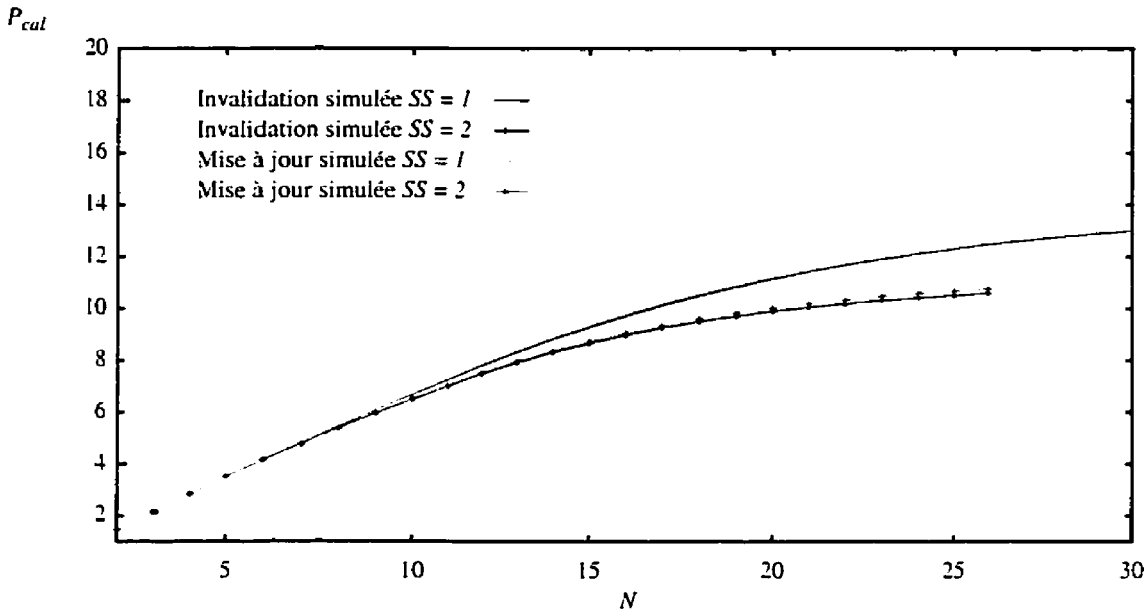


Figure 5.19 Effet du nombre de serveurs secondaires: $W = 0.9$

Recommandations

Il faut utiliser, le moins possible de serveurs secondaires, vu qu'ils détériorent les performances du système.

5.2.7 Effet de la vitesse du réseau

Pour voir l'effet de ce paramètre sur les performances globales de notre système réparti, nous augmentons le temps de service de notre système de référence de 0.0015 à 0.05 UT.

Les courbes obtenues sont représentées aux figures 5.20 et 5.21.

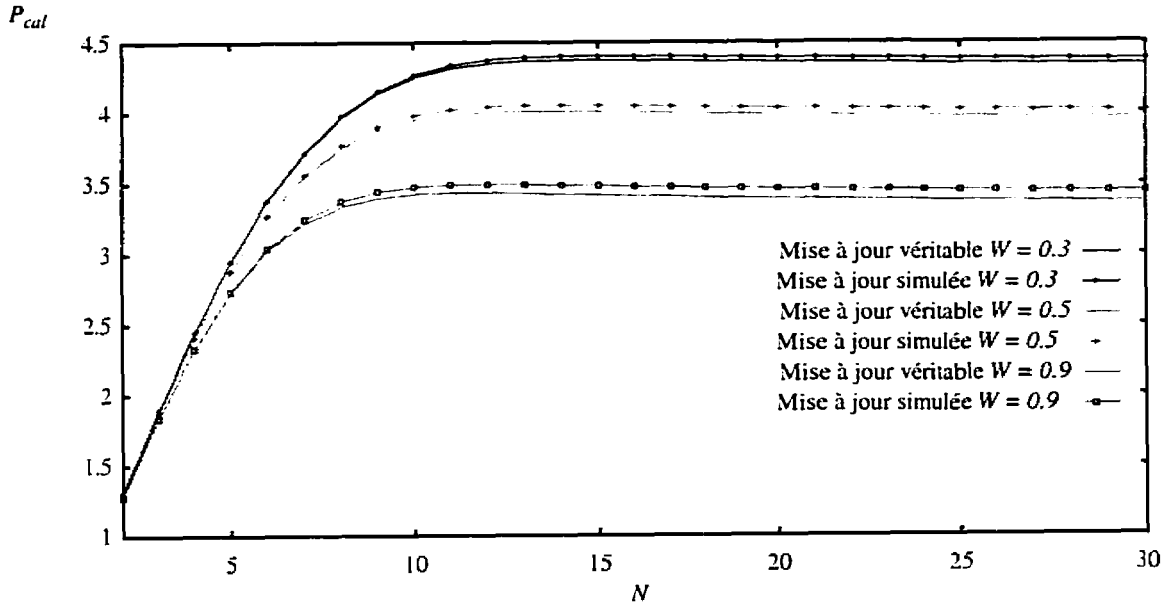


Figure 5.20 Effet de la vitesse du réseau sur le protocole de mise à jour: $T_{réseau} = 0.05$

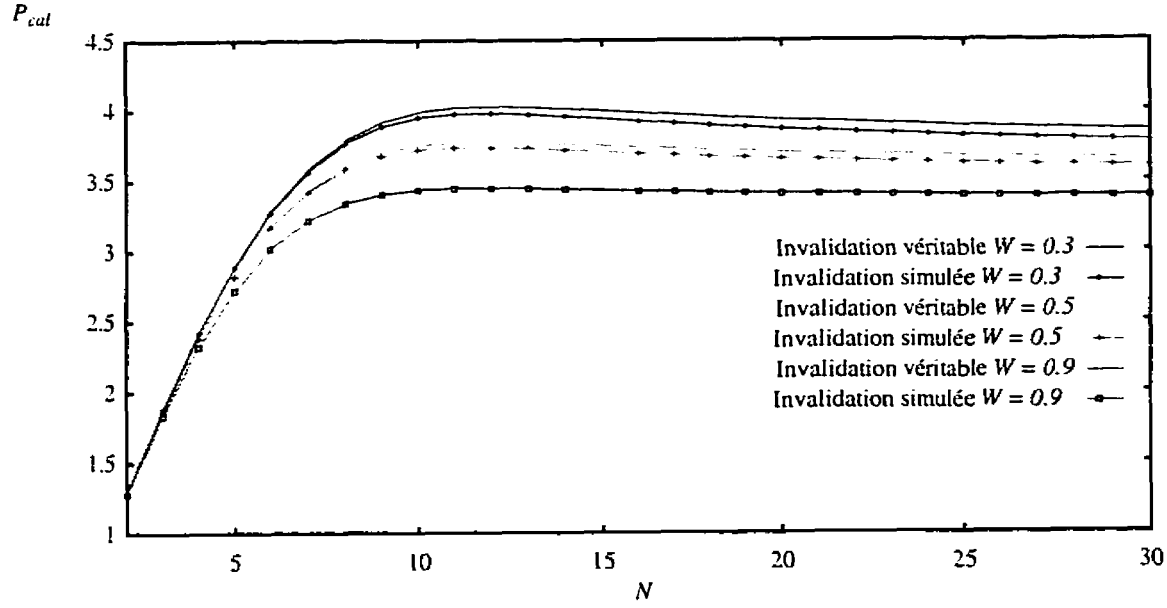


Figure 5.21 Effet de la vitesse du réseau sur le protocole d'invalidation: $T_{réseau} = 0.05$

On constate d'abord que les performances sont meilleures dans le cas d'une diffusion véritable. Ceci est dû au fait que le nombre de paquets transmis au réseau dans le cas d'une diffusion simulée est plus important que dans le cas d'une diffusion véritable. D'autre part, la puissance de calcul croît jusqu'à un certain seuil et commence à diminuer. Il s'agit de l'écroulement du système. Ceci s'explique par le fait que le réseau devient saturé et est incapable de servir les requêtes qui lui sont transmises par les postes de travail, les serveurs principaux et les serveurs secondaires.

Recommandations

Si le réseau est de type à diffusion, il est préférable d'utiliser la diffusion véritable. Dans le cas contraire, il faut utiliser une diffusion simulée. Dans les deux cas, il faut éviter la saturation du système en dimensionnant adéquatement la vitesse du réseau en fonction du nombre de postes. Autrement, l'ajout de postes suplémentaires n'entraine pas d'augmentation de la puissance de calcul.

CHAPITRE 6

CONCLUSION

Les protocoles d'invalidation en écriture et de mise à jour en écriture sont généralement utilisés dans les systèmes à DSM, afin de maintenir la cohérence entre les données partagées. Leurs fonctionnement font intervenir plusieurs paramètres interdépendants. Pour une application, Il est donc difficile, de déterminer parmi ces protocoles, celui qui donnera les meilleurs résultats, ainsi que d'évaluer le rôle et la contribution de chacun de ces paramètres sur la performance du système. Pour le faire, il faut disposer, d'un modèle simple qui permet d'évaluer instantanément les performances du système à analyser. Si ce modèle est bien conçu, il devra en plus, faciliter l'étude du rôle de chacun des paramètres du système afin de dégager les différents cas d'utilisation de chacun de ces protocoles.

Ce travail nous a d'abord permis d'analyser les paramètres et les algorithmes dont il faut tenir compte lors de la conception des systèmes DSM. Par la suite, nous avons obtenu les modèles analytiques des systèmes que nous avons spécifiés. Les résultats obtenus à l'aide de ces modèles ont été validés par des résultats fournis par des modèles de simulation.

Les modèles analytiques obtenus sont formés de trois composantes: le modèle de l'état de données, le modèle des copies des données, le modèle du processeur.

Le modèle de l'état des données: Il s'agit des chaînes de Markov qui modélisent les différents états par où transitent les données partagées dans chaque mémoire locale.

Le modèle des copies des données: Il s'agit des chaînes de Markov qui modélisent l'évolution du nombre de copies des données dans le système.

Le modèle du processeur: Il est constitué d'une seule chaîne de routage à plusieurs clients. Il décrit le système physique ainsi que les cheminements à suivre par chaque requête.

Les avantages de ces modèles analytiques sont qu'ils sont simples, qu'ils tiennent compte des différentes techniques de diffusion et qu'ils font intervenir tous les paramètres des systèmes physiques. D'autre part, les algorithmes pour la mesure de performance ont été implantés. Les programmes obtenus donnent un temps de réponse rapide sur un ordinateur personnel, quel que soit la taille du système. Il faut aussi noter que, ces programmes utilisent les paramètres du système comme entrées; ce qui rend facile l'étude du rôle et la contribution de chacun d'eux sur les performances du système.

Les résultats numériques que nous avons obtenus, et les études que nous avons réalisées à l'aide de ces programmes nous ont permis de tirer des conclusions quant aux paramètres du système.

Le temps d'interarrivée des accès aux variables: Lorsqu'il augmente, les performances du système s'améliorent. S'il devient très grand, les performances qu'on obtient sont identiques pour les deux protocoles.

La capacité des mémoires locales: Lorsque les tailles de mémoires locales augmentent, les performances du système s'améliorent si ce dernier est muni du protocole de mise à

jour en écriture. Ce gain de performance est d'autant plus important que le taux d'écriture est faible. Par contre, ce paramètres est presque sans effet sur les performances lorsque le système est muni du protocole d'invalidation en écriture.

Le nombre de serveurs principaux: L'augmentation du le nombre de serveurs principaux engendre un gain de performance. Cependant, il existe un seuil au delà duquel l'ajout d'un SP additionnel n'affecte pas les performances du système. La valeur de ce seuil est d'autant plus élevé que le taux d'écriture est grand.

Le nombre de serveurs secondaires: L'ajout d'un SS a pour effet de détériorer les performances du système. Leur nombre doit être le plus petit possible.

Les techniques de diffusion: Lorsque le réseau est très rapide, ces deux techniques donnent les mêmes performances. Par contre, lorsque le réseau est moins rapide, la diffusion simulée est à proscrire.

Il faut noter que les modèles que nous avons obtenus ainsi que les conclusions qui en découlent ne sont valables que pour les systèmes qui ont été spécifiés dans ce mémoire. Cependant, la démarche utilisée dans le cadre de ce mémoire peut être utilisée pour l'obtention du modèle analytique de tout système à DSM.

RÉFÉRENCES

[Krakowiak 87]

Krakowiack, S. (1987). *Principe des systèmes d'exploitation des ordinateurs*, Dunod, France.

[Bertekas 89]

Bertsekas, D. et Gallager, R. (1992). *Data Networks*, Prentice-Hall, New Jersey.

[Stenström 90]

Stenström, P. (1990) *A Survey of Cache Coherence Schemes for Multiprocessors*, IEEE Computer, juin 1990, pp. 12-24.

[Tanenbaum 85]

Tanenbaum, A.S. et Van Renesse, R. (1985). *Distributed Operating Systems*, Computing Surveys, Vol.17, No.4, décembre 1985.

[Nitzberg 91]

Nitzberg, B. et Virginia, L. (1991). *Distributed Shared Memory: A Survey of issues and Algorithms*, IEE Computer, août 1991, pp 52-60.

[Balter 91]

Balter, R., Banâtre, et J.-P., Krakowiack, S. (1991). *Construction des systèmes d'exploitation répartis*, INRIA, France.

[Lenoski 90]

Lenoski, D. et al. (1990). *The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor*, Proc. 17th. Int'l Symp. Computer Architecture, IEE Comp. Sc. press, Los Alamitos, Calif. Order No. 2047, 1990, pp. 148-159.

[Carriero 89]

Carriero, N. et Gelernter, D. (1989). *Linda in Context*, Communications of the ACM, Vol. 32, No. 4, avril 1989, pp 444-458.

[Bennett 90]

Bennet, J., Carter, J. et Zwaenepoel, W. (1990). *Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence*, Proc. 1990 Conf. Principles and Practice of Parallel Programming, ACM Press, New York, 1990, pp. 168-176.

[Bisiani 90]

Bisiani, R. et Ravishankar, M. (1990). *Plus: A Distributed shared-Memory System*, proc. Int'l Symp. Computer Architecture, IEE CS press, Los Alamitos, Calif., Order No. 2047, 1990, pp. 115-124.

[Tanenbaum 87]

Tanenbaum, A. (1987). *Les systèmes d'exploitation*, Prentice Hall, New Jersey.

[Archibal 86]

Archibal, J. et Baer, J.L. (1986). *Cache Coherence Protocols: Evaluation Using A Multiprocessor Simulation Model*, ACM Transaction on Computer Systems, Vol.4, No.4, novembre 1986, pp. 273-298.

[Vernon 86]

Vernon, M.K., et Holliday, M.A. (1986). *Performance Analysis of Multiprocessor Cache Coherence Protocols Using Generalized Timed Petri Nets*, Proc. ACM Sigmetrics Conf. 1986 pp. 9-17.

[Yang 88]

Yang, Q. et Bhuyan L.N. (1988). *A queueing Network Model for A Cache Coherence Protocol on Multiple-bus Multiprocessors*, Proc. Int'l. Conf. on Parallel Processing, août 1988, pp 130-137.

[Bhuyan 89]

Bhuyan L.N., Liu, B.-C. et Ahmed, I. (1989), *Analysis of MIN Based Multiprocessors With Private Cache Memories*, Proc. of the 1989 Int'l Conf. on Parallel Processing, août 1989, pp I-51-I-58.

[Bruel 80]

Bruell, S.C., et Balbo, G. (1980). *Computational Algorithms for Closed Queuing Networks*, Elsevier North Holland, New York.

[Reiser 79]

Reiser, M. (1979). *A queuing network analysis of computer communication networks with window flow control*, IEE Tran. on Commun. Vol. COM-27, août 1979, pp 1199-1209.

[Kleinrock 75]

Kleinrock, L. (1975). *Queueing Systems*, Vol. 1, New York, Wiley.

[Karian 90]

Karian Z.A., et Dudewicz, E.J. (1990). *Modern Satistical, Systems, and GPSS Simulation*, Computer Science Press, New York.

[Schriber 90]

Schriber, T. J., 1990. *Simulation Using GPSS*, Krieger Publishing Company, Malabar, Florida.

[Jacobson 82]

Jacobson P.A. et Lazowska, E.D. (1988). Analysing Queueing Networks with Simultaneous Ressource Possession, Comm. of the ACM février 1982, Vol 25, No. 2, pp 142-151.

[Lazowska 86]

Lazowska, E.D. et Sevcik, K.C. (1986). *Computer System Performance Evaluation Using Queueing network Models*, Ann. Rev. Comput. Sci. 1986, pp. 107-137.

ANNEXE 1

RÉSULTATS NUMÉRIQUES

Cet annexe comprend les résultats numériques obtenus lors de la validation des modèles. Ces résultats sont présentés sous forme de tableau. Dans la première colonne de chaque tableau, sont indiqués les différents taux d'écriture utilisés. Dans les trois colonnes suivantes, sont reportées les mesures de performances obtenues dans le cas d'un système faiblement chargé; dans les trois dernières, sont indiquées les performances dans le cas d'un système fortement chargé. Les résultats contenus dans les trois premiers tableaux concernent le protocole de mise à jour en écriture et dans les trois derniers, le protocole d'invalidation en écriture.

Tableau I: Résultats numériques pour une mise à jour simulée

Taux d'écriture W	Système #1 P_{ana}	Système #1 P_{sim}	Système #1 *Écart* x10^3	Système #2 P_{ana}	Système #2 P_{sim}	Système #2 *Écart* x10^3
0.0	2.9052	2.9065	1.3	2.5811	2.5710	-10.1
0.1	2.9023	2.9043	2.0	2.5421	2.5340	-8.1
0.2	2.8997	2.9022	2.5	2.5024	2.4987	-3.7
0.3	2.8973	2.9001	2.8	2.4602	2.4574	-2.8
0.4	2.8950	2.8979	2.9	2.4147	2.4133	-1.4
0.5	2.8927	2.8957	3.0	2.3656	2.3689	3.3
0.6	2.8905	2.8933	2.8	2.3128	2.3171	4.3
0.7	2.8883	2.8908	2.5	2.2564	2.2570	0.6
0.8	2.8860	2.8879	1.9	2.1970	2.2029	5.9
0.9	2.8838	2.8848	1.0	2.1353	2.1373	2.0
1.0	2.8816	2.8815	-0.1	2.0721	2.0665	-5.6

Tableau II: Résultats numériques pour une mise à jour véritable

Taux d'écriture W	Système #1 P_{ana}	Système #1 P_{sim}	Système #1 *Écart* x10^3	Système #2 P_{ana}	Système #2 P_{sim}	Système #2 *Écart* x10^3
0.0	2.9053	2.9066	1.3	2.6136	2.6007	-12.9
0.1	2.9025	2.9043	1.8	2.5901	2.5840	-6.1
0.2	2.9000	2.9023	2.3	2.5682	2.5639	-4.3
0.3	2.8977	2.9003	2.6	2.5467	2.5434	-3.3
0.4	2.8956	2.8980	2.4	2.5253	2.5209	-4.4
0.5	2.8935	2.8959	2.4	2.5035	2.5005	-3
0.6	2.8914	2.8934	2.0	2.4813	2.4780	-3.3
0.7	2.8893	2.8909	1.6	2.4585	2.4530	-5.5
0.8	2.8873	2.8880	0.7	2.4351	2.4297	-5.4
0.9	2.8857	2.8848	-0.5	2.4111	2.4039	-7.2
1.0	2.8832	2.8815	-1.7	2.3863	2.3773	-9.0

Tableau III: Résultats numériques pour une mise à jour par chaînage

Taux d'écriture W	Système #1 P_{ana}	Système #1 P_{sim}	Système #1 *Écart* x10^3	Système #2 P_{ana}	Système #2 P_{sim}	Système #2 *Écart* x10^3
0.0	2.9052	2.9067	1.5	2.5811	2.5792	-1.9
0.1	2.9023	2.9043	2.0	2.5421	2.5495	7.4
0.2	2.8997	2.9023	2.6	2.5024	2.5212	18.8
0.3	2.8973	2.9002	2.9	2.4602	2.4894	29.2
0.4	2.8950	2.8981	3.1	2.4147	2.4565	41.8
0.5	2.8927	2.8959	3.2	2.3656	2.4201	54.5
0.6	2.8905	2.8935	3.0	2.3128	2.3746	61.8
0.7	2.8883	2.8909	2.6	2.2564	2.3297	73.3
0.8	2.8860	2.8881	2.1	2.1970	2.2783	81.3
0.9	2.8838	2.8849	1.1	2.1353	2.2249	89.6
1.0	2.8816	2.8815	-0.1	2.0721	2.1656	93.5

Tableau IV: Résultats numériques pour une invalidation simulée

Taux d'écriture W	Système #1 P_{ana}	Système #1 P_{sim}	Système #1 *Écart* x10^3	Système #2 P_{ana}	Système #2 P_{sim}	Système #2 *Écart* x10^3
0.0	2.9052	2.9065	1.3	2.5811	2.5707	-10.4
0.1	2.8996	2.9013	1.7	2.5207	2.5176	-3.1
0.2	2.8955	2.8972	1.7	2.4689	2.4719	3.0
0.3	2.8924	2.8936	1.2	2.4221	2.4289	6.8
0.4	2.8800	2.8908	0.8	2.3785	2.3881	9.6
[illegible]	2.8882	2.8885	0.3	2.3368	2.3469	10.0
0.6	2.8867	2.8866	-0.1	2.2962	2.3127	16.5
0.7	2.8856	2.8851	-0.5	2.2561	2.2759	19.8
0.8	2.8846	2.8837	-0.9	2.2161	2.2349	18.8
0.9	2.8839	2.8825	-1.4	2.1760	2.2008	24.8
1.0	2.8833	2.8814	-1.9	2.1354	2.1633	27.9

Tableau V: Résultats numériques pour une invalidation véritable

Taux d'écriture W	Système #1 P_{ana}	Système #1 P_{sim}	Système #1 *Écart* x10^3	Système #2 P_{ana}	Système #2 P_{sim}	Système #2 *Écart* x10^3
0.0	2.9053	2.9067	1.4	2.6136	2.6014	-12.2
0.1	2.8997	2.9014	1.7	2.5704	2.5688	-1.6
0.2	2.8957	2.8973	1.6	2.5373	2.5401	2.8
0.3	2.8928	2.8939	1.1	2.5109	2.5179	7.0
0.4	2.8906	2.8910	0.4	2.4891	2.4951	6.0
0.5	2.8889	2.8889	0.0	2.4708	2.4771	6.3
0.6	2.8876	2.8868	-0.8	2.4550	2.4593	4.3
0.7	2.8866	2.8852	-1.4	2.4412	2.4449	3.7
0.8	2.8859	2.8838	-2.1	2.4289	2.4320	3.1
0.9	2.8854	2.8826	-2.8	2.4177	2.4188	1.1
1.0	2.8849	2.8815	-3.4	2.4073	2.4040	-3.3

Tableau VI: Résultats numériques pour une invalidation par chaînage

Taux d'écriture W	Système #1 P_{ana}	Système #1 P_{sim}	Système #1 *Écart* x10^3	Système #2 P_{ana}	Système #2 P_{sim}	Système #2 *Écart* x10^3
0.0	2.9052	2.9066	1.4	2.5811	2.5758	-5.3
0.1	2.8996	2.9015	1.9	2.5207	2.5358	15.1
0.2	2.8955	2.8972	1.7	2.4689	2.4979	29.0
0.3	2.8924	2.8938	1.4	2.4221	2.4645	42.4
0.4	2.8800	2.8910	1.0	2.3785	2.4344	55.9
0.5	2.8882	2.8886	0.4	2.3368	2.3061	69.2
0.6	2.8867	2.8867	0.0	2.2962	2.3754	79.2
0.7	2.8856	2.8852	-0.4	2.2561	2.3436	87.5
0.8	2.8846	2.8838	-0.8	2.2161	2.3122	96.1
0.9	2.8839	2.8826	-1.3	2.1760	2.2793	103.3
1.0	2.8833	2.8815	-1.8	2.1354	2.2431	107.7

ANNEXE 2
PROGRAMMES DES MODÈLES ANALYTIQUES ET DE SIMULATION

Les programmes sous forme texte que nous avons utilisé pour obtenir les résultats numériques de nos modèles sont contenus dans la disquette qui accompagne ce mémoire. Leurs descriptions se trouvent au tableau VII ci-dessous.

Les programmes dont les noms ont une extension C peuvent être compilés avec n'importe quel compilateur C. Les programmes dont les noms ont une extension GPS peuvent être compilés avec n'importe quel compilateur GPSS.

Tableau VII: Description des programmes

Fichiers	Protocoles	Techniques de diffusion	Langages	Modèle
IS.C	Invalidation	Simulée	C	Analytique
IV.C	Invalidation	Véritable	C	Analytique
MS.C	Mise à jour	Simulée	C	Analytique
MV.C	Mise à jour	Véritable	C	Analytique
IS.GPS	Invalidation	Simulée	GPSS	Simulation
IV.GPS	Invalidation	Véritable	GPSS	Simulation
IC.GPS	Invalidation	Chaînage	GPSS	Simulation
MS.GPS	Mise à jour	Simulée	GPSS	Simulation
MV.GPS	Mise à jour	Véritable	GPSS	Simulation
MC.GPS	Mise à jour	Chaînage	GPSS	Simulation

Une disquette accompagne ce mémoire de maîtrise.

Toute personne intéressée à se la procurer doit contacter:

Ecole Polytechnique de Montréal

Prêt entre bibliothèques

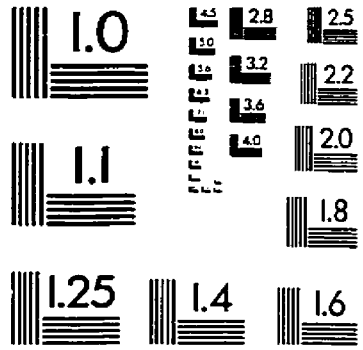
B.P. 6079, Succursale Centre-Ville

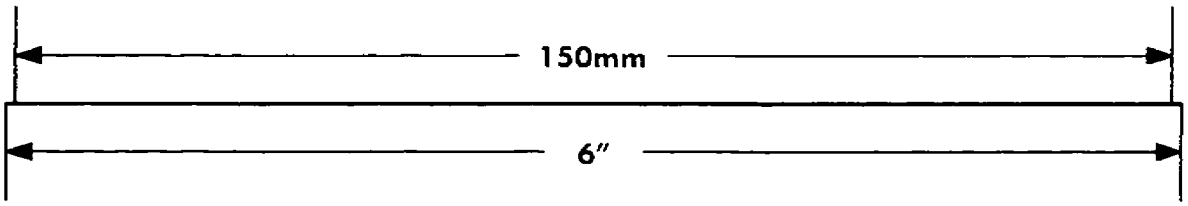
Montréal, Québec

H3C 3A7

Tél: (514) 340-4846
Télécopieur: (514) 340-4026

TEST TARGET (QA-3)





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989